FR Family MB2198-01 Emulator System Getting Started Guide

Doc. No. 002-05222 Rev. *A

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
http://www.cypress.com

Copyrights

Trademarks

Source Code

Disclaimer

Contents

1. Introduction

This document will help you how to debug an emulation system with the MB2198-01 Emulation with the Softune Workbench V60L06. For in-depth information please refer to the following manuals:

- MB2198-01 Hardware Manual (Emulator)
- MB2198-01 Getting Started Application Note (MCU-AN-391027)
- MB2198-01 Installation Guide Application Note (MCU-AN-391026)

This document describes the debugging methods of a MB91V460 system together with a SK-91F467-Flexray target board. Please note, that the debugging principle is the same for other systems.

2. Sample Program

Sample Program for Debugging

2.1 Start new project with Softune Workbench

At first choose an evaluation MCU (here: MB91467D), copy the template project of the “Softune samples” into an own folder (here: “Emulation_Test”) and start the Softune Workbench Software.

2.2 “Main.c”

The following sample code program, based on the standard template project, is used for demonstrating emulation and debugging. Please change “Main.c” to the following:

```
/************************@INCLUDE_START************************/
#include "mb91467d.h"
#include "vectors.h"
/************************@INCLUDE_END************************/

/******************@FUNCTION_DECLARATION_START*****************/
void wait (short int cnt)
{
    int i;
    PDR16 = 0xFF;
    for(i=0;i<cnt;i++);
    PDR16 =0x00;

}

void main(void)
{
    __EI();                     /* enable interrupts */
    __set_il(31);               /* allow all levels */
    InitIrqLevels();            /* init interrupts */

    PORTEN = 0x3;               /* enable I/O Ports */
                                /* This feature is not supported by MB91V460A */
                                /* For all other devices the I/O Ports must be enabled*/
    PFR16 = 0x00;
    DDR16 = 0xFF;

    while(1)                    /* endless loop */
    {
       HWWD_CL = 0;
       __asm(" NOP");
       wait (5000);
       __asm(" NOP");
    }
}
```

This program is only an example with no “great assignment“. It contains a simple wait-function (`void wait`) , which needs an short integer value for the wait time. The resulting delay time depends on the value itself and the clock speed of the emulation system.

At first the interrupts are enabled (although they are not used in this example), then the Port16 of the MCU is set to “output“ (Port16 is the LED-Port of the SK-91F467-Flexray board). Then the `wait` function is called with the value `5000`.

2.3 Compiling “Main.c”

To compile the project, please use “Setup Project” first. In Project→Setup Project→C Compiler→Category: Optimize has to be selected General-purpose Optimization Level: **None**.

Then compile the project by clicking on  Build all source files regardless of data or selecting *Project→Build*, or pressing “Ctrl-F8”.

Watch for error messages. If all is ok, you will get the following message:

Now building...

----------Configuration: 91460_template_91467D.prj - STANDALONE----------

vectors.c

Start91460.asm

mb91467d.asm

MAIN.c

Now linking...

<your path>91460_template_91467d\STANDALONE\ABS\91460_template_91467d.abs

Now starting load module converter...

<your path>\91460_template_91467d\STANDALONE\ABS\91460_template_91467d.mhx

----

No Error.

----

3. Debugging, First Steps

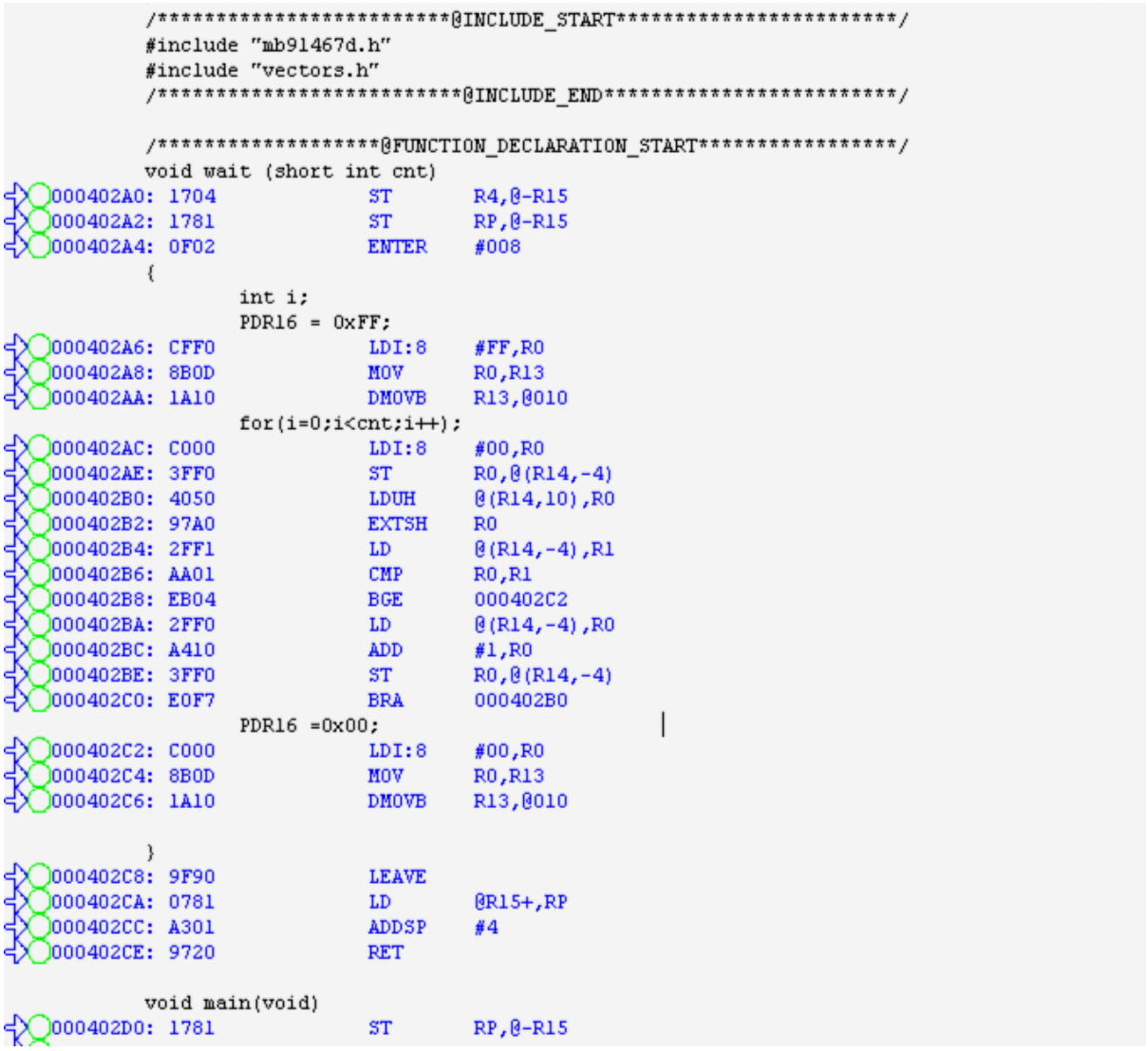
How to Enter Debugging Mode

3.1 Setup Hardware

For the next steps you have to set up your emulation hardware. Please refer to the application notes “Installation Guide MB2198-01” (MCU-AN-391026) and “Emulator System MB2198-01, Getting started” (MCU-AN-391027) for details.

3.2 Entering Debugger Mode

After successful compilation of the project start the debugging mode via COM1/2, USB or LAN by double clicking on the regarding Debug-“.sup”-entry in the workspace window. After successful connection to the emulator, reset MCU, open “Main.c” (close it first, if it is open) and then click on right mouse button and select “*Mix Display*”. Your generated code should look like the following sample code.



```
            /*************************@INCLUDE_START************************/
            #include "mb91467d.h"
            #include "vectors.h"
            /*************************@INCLUDE_END************************/

            /*******************@FUNCTION_DECLARATION_START*****************/
            void wait (short int cnt)
000402A0: 1704              ST      R4,@-R15
000402A2: 1781              ST      RP,@-R15
000402A4: 0F02              ENTER   #008
            {
                    int i;
                    PDR16 = 0xFF;
000402A6: CFF0              LDI:8   #FF,R0
000402A8: 8B0D              MOV     R0,R13
000402AA: 1A10              DMOVB   R13,@010
                    for(i=0;i<cnt;i++);
000402AC: C000              LDI:8   #00,R0
000402AE: 3FF0              ST      R0,@(R14,-4)
000402B0: 4050              LDUH    @(R14,10),R0
000402B2: 97A0              EXTSH   R0
000402B4: 2FF1              LD      @(R14,-4),R1
000402B6: AA01              CMP     R0,R1
000402B8: EB04              BGE     000402C2
000402BA: 2FF0              LD      @(R14,-4),R0
000402BC: A410              ADD     #1,R0
000402BE: 3FF0              ST      R0,@(R14,-4)
000402C0: E0F7              BRA     000402B0
                    PDR16 =0x00;
000402C2: C000              LDI:8   #00,R0
000402C4: 8B0D              MOV     R0,R13
000402C6: 1A10              DMOVB   R13,@010

            }
000402C8: 9F90              LEAVE
000402CA: 0781              LD      @R15+,RP
000402CC: A301              ADDSP   #4
000402CE: 9720              RET

            void main(void)
000402D0: 1781              ST      RP,@-R15
```

```
000402D2: 0F01            ENTER    #004
          {
              __EI();                  /* enable interrupts */
000402D4: 9310            ORCCR    #10
              __set_il(31);            /* allow all levels */
000402D6: 871F            STILM    #1F
              InitIrqLevels();         /* init interrupts */
000402D8: 9F8C00040000    LDI:32   #00040000,R12
000402DE: 971C            CALL     @R12

              PORTEN = 0x3;            /* enable I/O Ports */
000402E0: C030            LDI:8    #03,R0
000402E2: 9F8C00000498    LDI:32   #00000498,R12
000402E8: 16C0            STB      R0,@R12
                                       /* This feature is not supported by MB91V460A */
                                       /* For all other devices the I/O Ports must be enabled*/
                   PFR16 = 0x00;
000402EA: C000            LDI:8    #00,R0
000402EC: 9F8C00000D90    LDI:32   #00000D90,R12
000402F2: 16C0            STB      R0,@R12
                   DDR16 = 0xFF;
000402F4: CFF0            LDI:8    #FF,R0
000402F6: 9F8C00000D50    LDI:32   #00000D50,R12
000402FC: 16C0            STB      R0,@R12

              while(1)                 /* endless loop */
              {
                 HWWD_CL = 0;
000402FE: 9F80000004C7    LDI:32   #000004C7,R0
00040304: 8070            BANDL    #7,@R0
                 __asm(" NOP");
00040306: 9FA0            NOP
                 wait (5000);
00040308: 9B041388        LDI:20   #01388,R4
0004030C: D7C9            CALL     \wait
                 __asm(" NOP");
0004030E: 9FA0            NOP
00040310: E0F6            BRA      000402FE
              }
          }
00040312: 9F90            LEAVE
00040314: 0081            LD       @(R13,R8),R1
00040316: 9720            RET


        /*******************@FUNCTION_DECLARATION_END*****************/
```

3.3 Using Bookmarks

Since Softune version V60L06 it is possible to set bookmarks in source code windows.

Bookmarked lines are marked with completely in green in source code lines. With the bookmark arrow buttons it can be stepped through the code, stopping at bookmarked lines.

3.4 Start Execution

To enter the run mode, click on  Run continuously or select *Debug→Run→Go*, or press “F5”.

Now the program is being executed. If you are using a target system with LEDs on Port16, you will see the LEDs flicker if SK-91F467D-Flexray Target board is used.

3.5 Stop Execution

To stop the MCU, click on  Stop execution or select *Debug→Abort*.

Now the system is halted, but it can be continued again by clicking on “*Run continuously*” or selecting “*Go*”.

3.6 Reset MCU

To reset the MCU, clock on  Reset MCU or select *Debug→Reset of MCU*.

Note: This works only if the application is stopped (e.g. breakpoint, etc.)

4. Monitoring and Manipulating

How to Monitor and Manipulate CPU Registers, Variables and Memory

4.1 Monitoring and Manipulating Processor Status

The Condition Code Register (CCR) is always displayed below the workspace window.

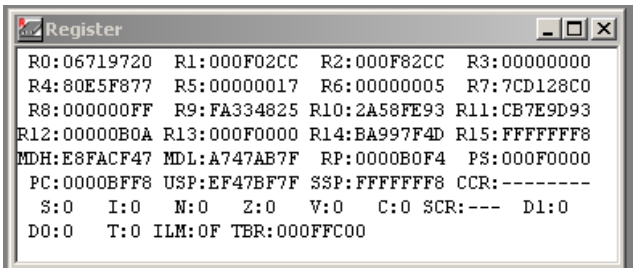
The flags are:

Abbr.	Flag Names
I	Interrupt enable flag (1 = enable)
S	Stack flag (0 = User stack; 1 = System stack)
T	Sticky bit flag (1 = shift right instruction executed)
N	Negative flag (MSB = 1 in last operation)
Z	Zero flag (Last operation resulted in “0”)
V	Overflow flag (Overflow at last operation)
C	Carry flag (Last operation caused carry)

The value of the flags can be easily changed by clicking into the white square. A “check mark” (√) indicates that the flag is set (== 1).

4.2 Monitoring and Manipulating CPU Registers

To display the CPU Registers window choose in the debugging mode: *View -> Register*. A new window will occur and look like this:



The registers are:

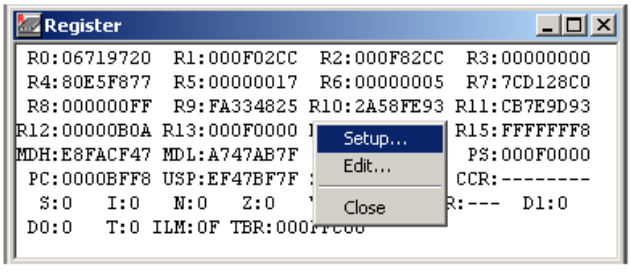
Abbr.	Flag Names
R0-R12	General purpose registers
R13	Virtual Accumulator
R14	Frame pointer
R15	Stack pointer
PC	Program counter
PS	Program status
TBR	Time Base register
RP	Return pointer
SSP	System stack pointer
USP	User stack pointer
MDH, MDL	Multiply-Divide register
TBR	Interrupt vector Table base register

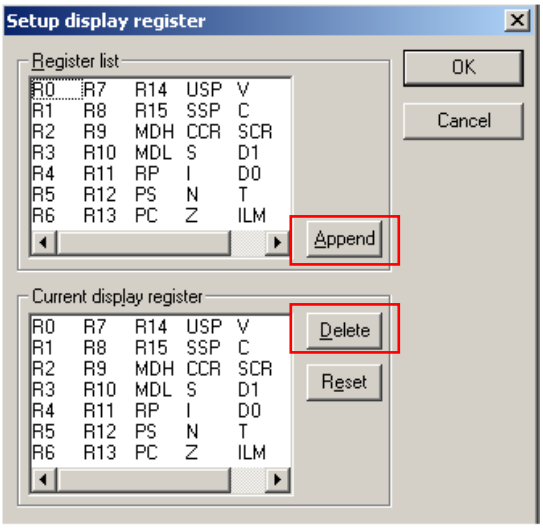
The contents of these registers can be changed by double-clicking them. A pop-up window will occur and look like the following picture:

Under *Register value* option one can enter a new value for the register. Note, that the values always are shown in hexadecimal notation by set-up default, but one can enter even decimal values (beginning with “D’”), binary values (beginning with “B’”), or octal values

(beginning with “O’”).

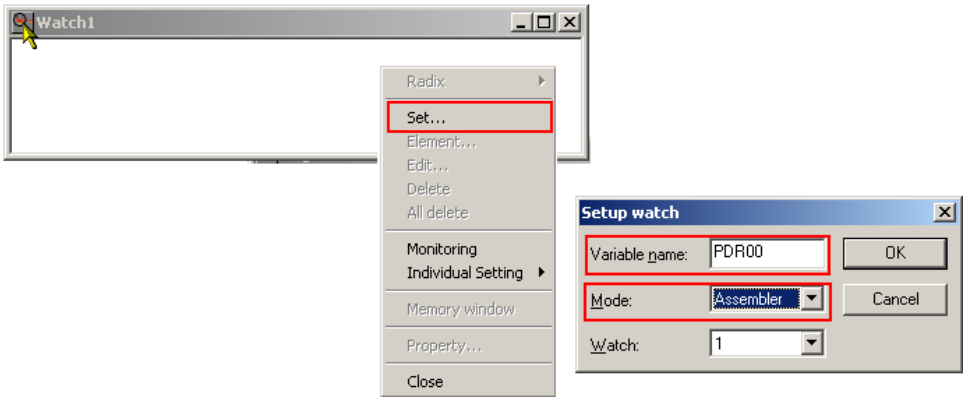
Registers can be added or removed by right clicking on *Register* window and selecting *setup…*



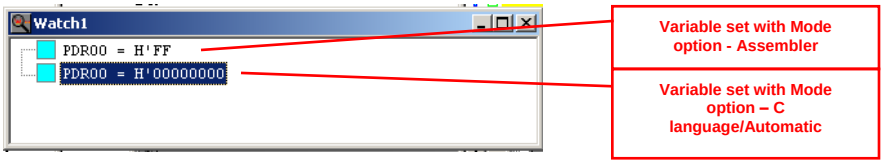


4.3 Monitoring and Manipulating Assembly Variables

To display assembly variables choose in the debugging mode: *View -> Watch -> Watch1*. A new window *Watch* will occur. Click in this window on the right mouse button and select Set… A pop-up window *Setup watch* will appear.



Under *Variable name* option one can enter the variable name of the assembly program. The Mode must be *Assembler* in this case. The Watch window will then contain the variable name and value. If we select other than this then watch window will show variables memory location and not the data contained at that location.

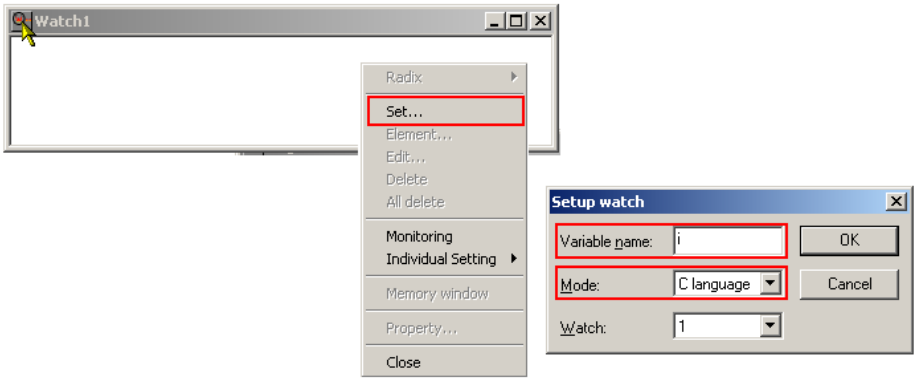


Note: You can change the radix of the value by right-clicking on the variable entry and choose via Radix: Binary, Octal, Decimal, or Hexadecimal.

To manipulate the value just double-click on the entry and enter in the pop-up window *Edit variable* a new value. The radix can be chosen via “D’”, “H’”, “B’”, or “O’”.

4.4 Monitoring and Manipulating C Variables

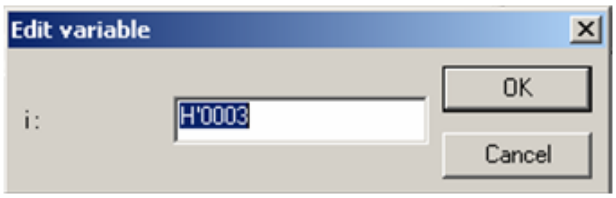
To display C variables choose in the debugging mode: *View -> Watch -> Watch1*. A new window *Watch* will occur. Click in this window on the right mouse button and select *Set…* A pop-up window *Setup watch* will appear.



Under *Variable name* option one can enter the variable name of the C program. The Mode must be *C language or Automatic* in this case. The Watch window will then contain the variable name and value.

Note: You can change the radix of the value by right-clicking on the variable entry and choose via Radix: Binary, Octal, Decimal, or Hexadecimal.

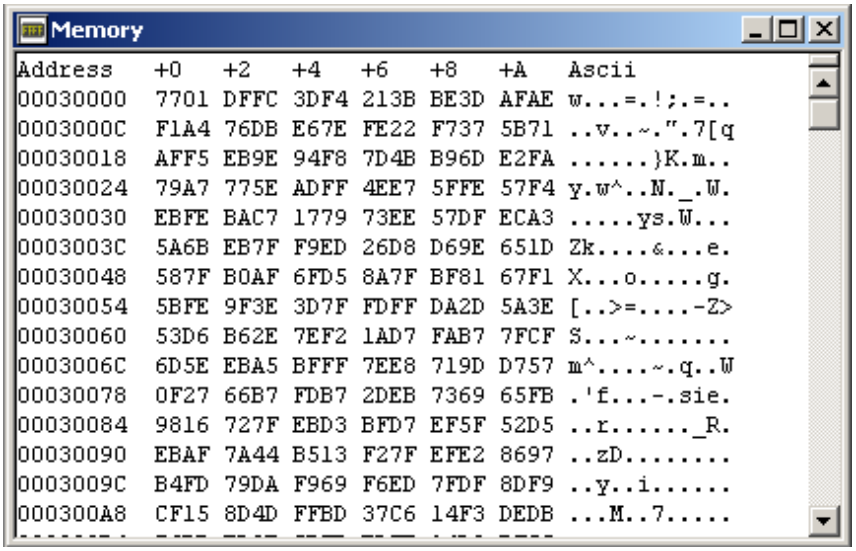
To manipulate the value just double-click on the entry and enter in the pop-up window *Edit variable* a new value. The radix can be chosen via “D’”, “H’”, “B’”, or “O’”.



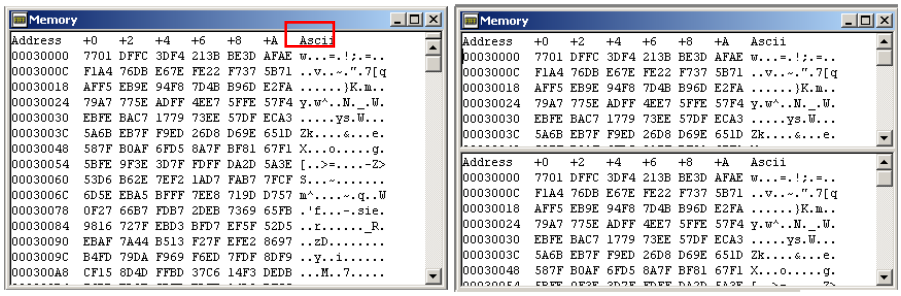
As explained in previous chapter to view memory content of Special Function Register, one should select *Assembler* under *Mode* option.

4.5 Monitoring and Manipulating Memory

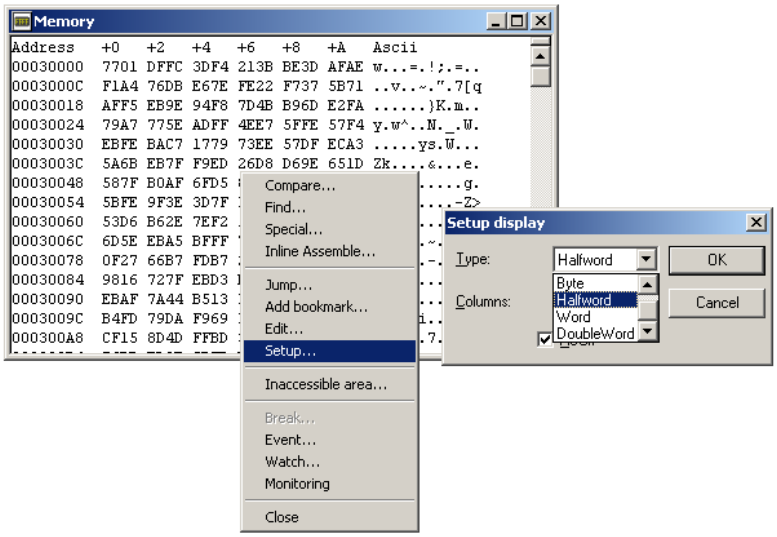
To display the MCU memory choose in the debugging mode: *View -> Memory*. A pop-up window *Memory* will occur and ask for the start address to be displayed. Type for instance H'2530 (or just 2530) for the RAM area. Following window occur which is something like a "Hex-Editor":



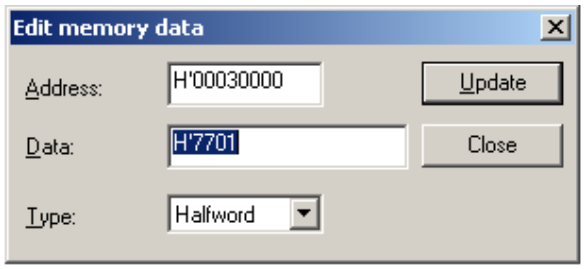
Memory window can be split. Move pointer near to top right corner, pointer will change to  ; drag it down with right mouse button pressed to split.



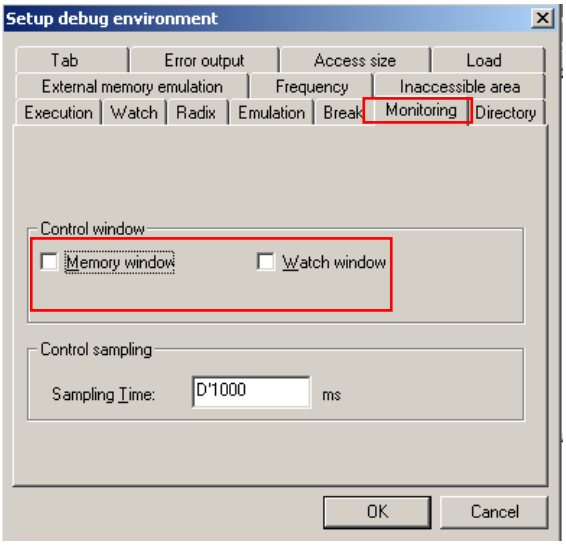
Also view can be set up to see data in bit, byte, word or long. Right click on memory window, click on setup. On a *Setup Display* dialog box select bit, byte, word or long from drop down menu.



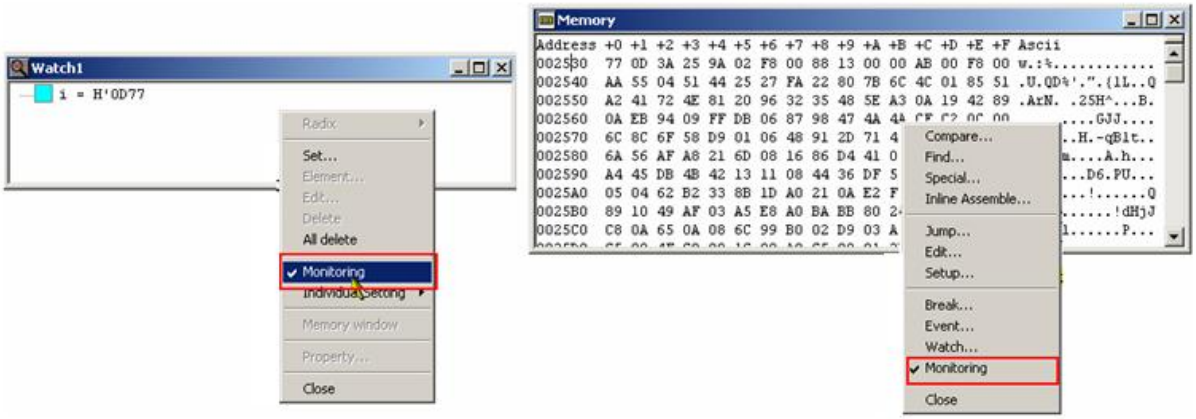
To change a memory content just double click on the respecting byte and *Edit memory data* dialog window pops up. In this window one can specify the address (default is the address of the clicked byte) and the new value. The value can be entered in hexadecimal, decimal, binary or octal format.



To see the memory in “real time” during execution, choose *Setup -> Debug environment -> Debug environment...* Then select the *Monitoring* tab and enter “D’1000” at *Sampling Time* and under *Control window* option click on check box *Memory window* and *Watch window*



Now, when the program is executed, *Watch* and *Memory* window is updated at 1000 ms, and one can see the addresses H'2530 in *Memory* window and variable 'i' in *Watch* window changing its values. Alternately one can select the same functionality by right clicking on *Watch/Memory* window, and on popup menu clicking on *Monitoring*

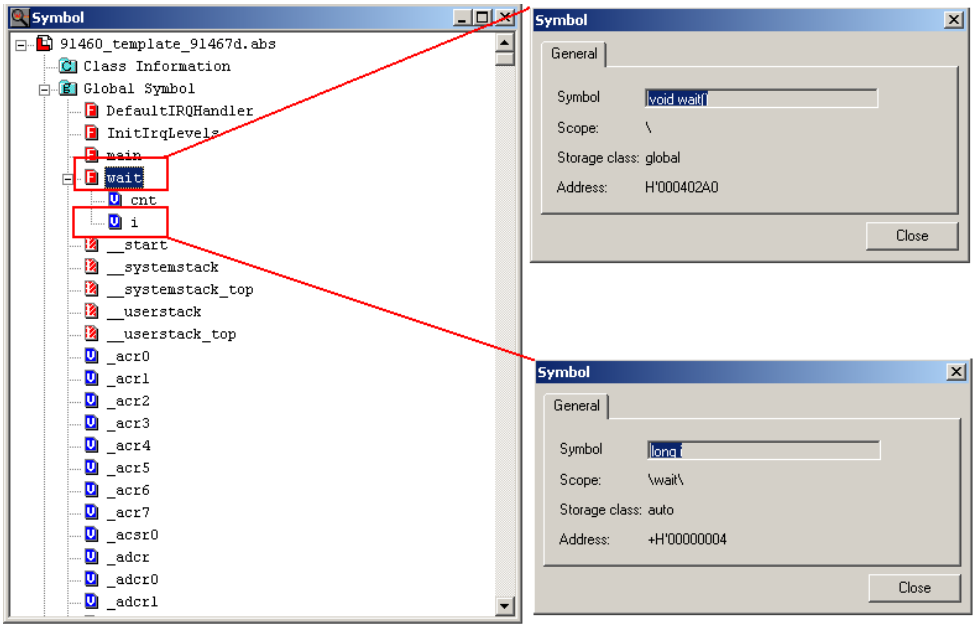


4.6 Symbol view

If one want to know where a variable is located in the memory than one can choose

View -> Symbol. Then unfold the sub list Project_name.abs/Global Symbol. Click with the right mouse button to the variable you want to get information about and select *Property…* .

The Symbol sub window shows then the address:



Icon Reference

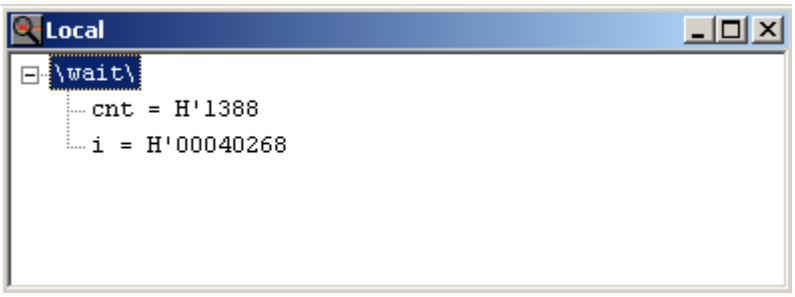
The following icons are used:

Icons	Flag Names
	Function
	Variable
	Label

4.7 Local variables

Local variables of functions can be displayed via View ->Local. A new window will open.

Note, that this window only shows contents if the debugger is in stop mode (e.g. breakpoint reached) and the actual function has local variables.



5. Breakpoints

How to Set Break Points

Code Break Point:

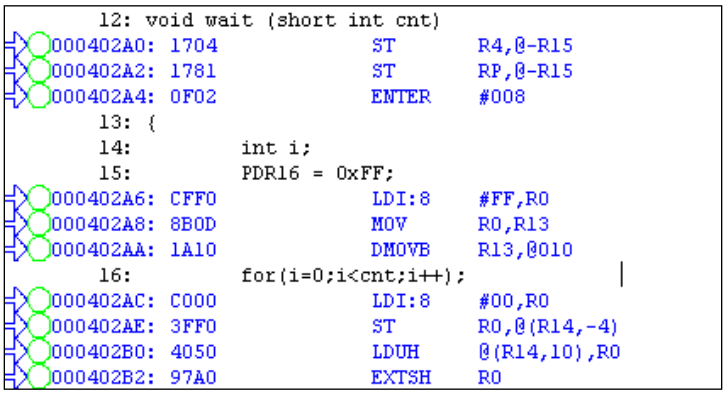
When code break point is set, program execution stops when the PC passes the break address (when instruction at that address is executed).

Four hardware code break points can be set and 4096 software break points can be set. For software break point, program halts every time when PC passes through the set address.

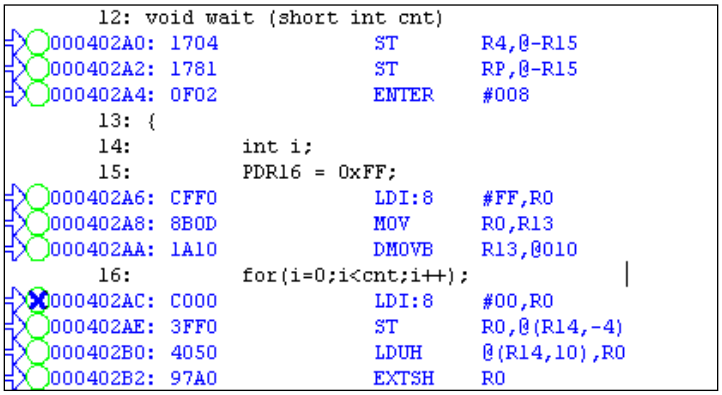
5.1 Setting Break points

5.1.1 Setting code break point through editor window

Each assembler line in the mixed mode display of the source code has a blue arrow and a green circle symbol



In these lines clicking into the circle can set a breakpoint or right click into the circle, click on Break Point Set/Reset. The symbol then turns to

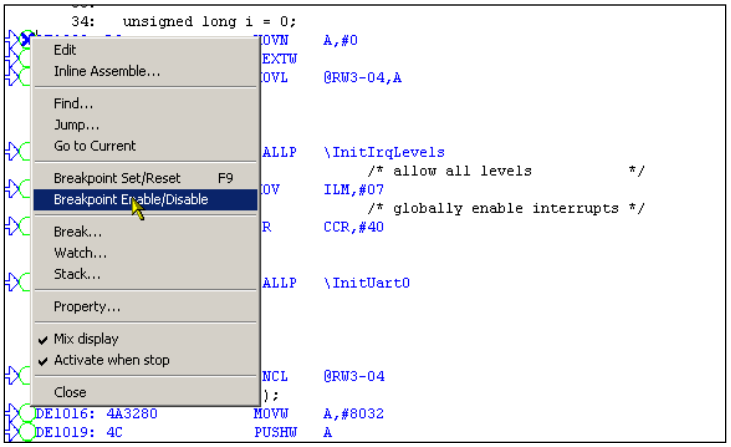


Depending on the selected break point type, a differently colored cross is shown in the circle. For the software break point (default), the cross is blue, while there is a red cross in the circle for the hardware break point

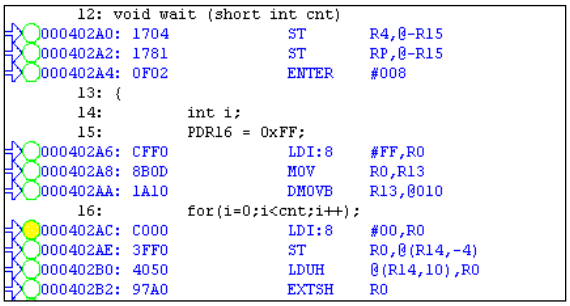
If you now start the execution, the CPU will halt on this break point. The actual line gets a yellow background color.

Clicking to this circle again or right clicking into the circle and clicking on Break Point Set/Reset, releases the break point.

Alternately, break point can be disabled by right clicking and on popup menu clicking Break Point enable/Disable.



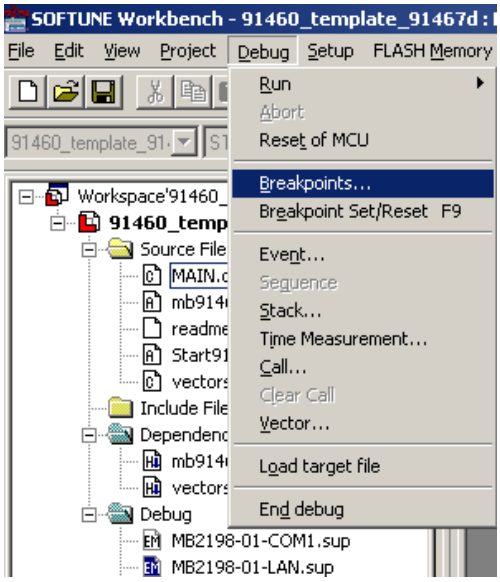
The symbol will change to



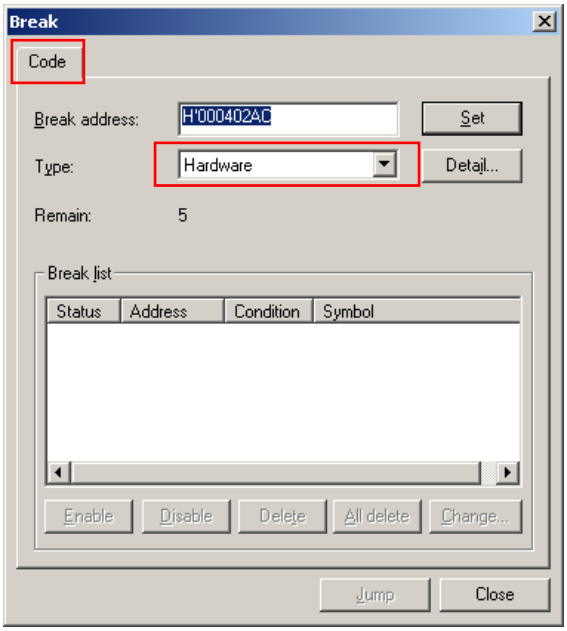
5.1.2 Setting code break point using Dialog box

A code break point can also be set using Dialog box

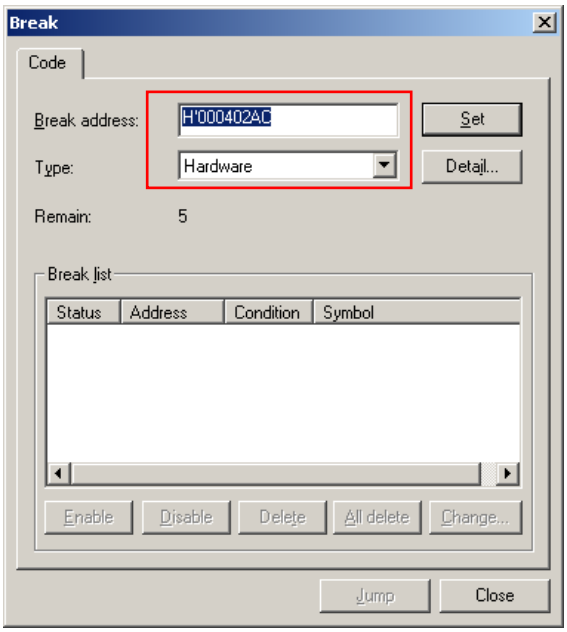
Open break point dialog box by clicking on *Breakpoints…* on Debug menu.



On *code* tab, select *software* or *Hardware* type breakpoint.

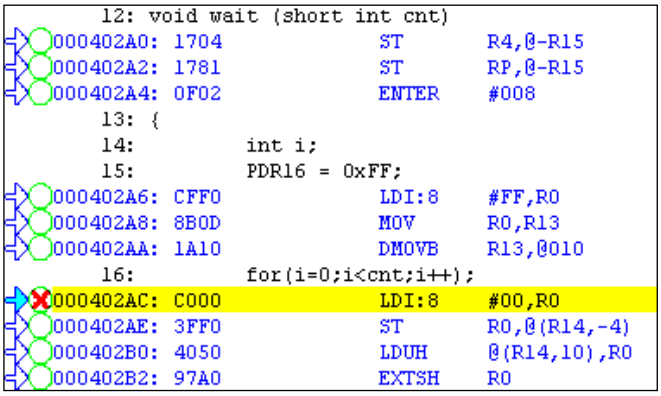


Type desired *Break address*

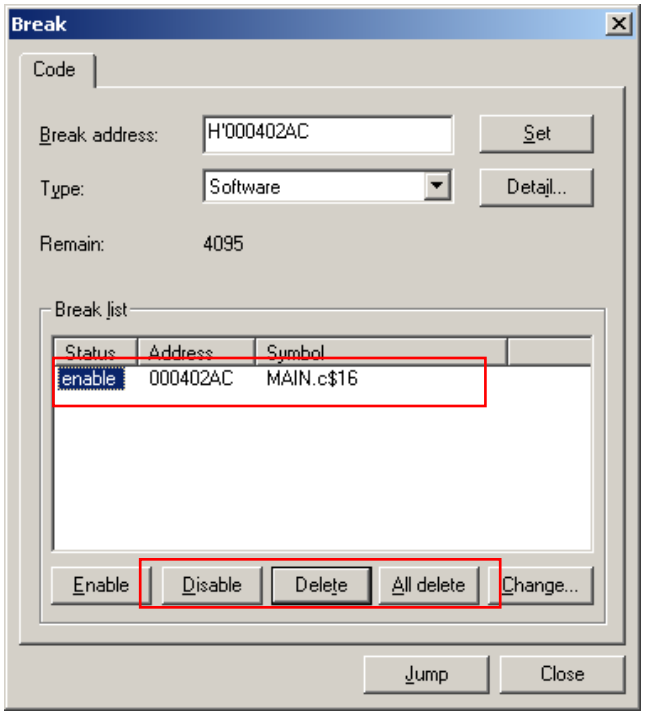


Click on *set,* to set software or Hardware breakpoint at required address

If you now start the execution, the CPU will halt on this break point. The actual line gets a yellow background color.



Breakpoint can be released by selecting breakpoint from *Break list,* and clicking on *Delete* button, alternately it can also be disabled by clicking *Disable* button

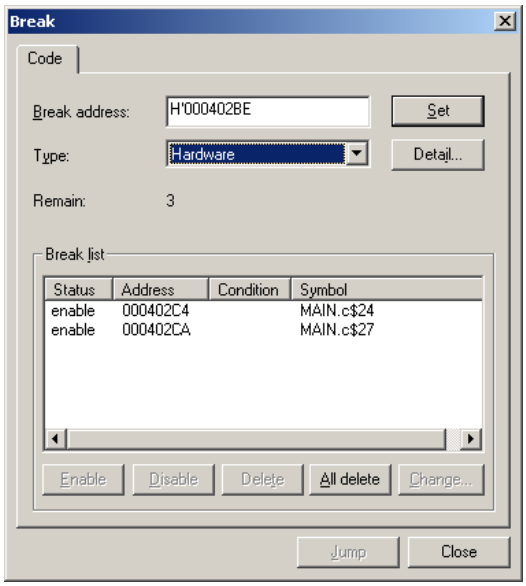


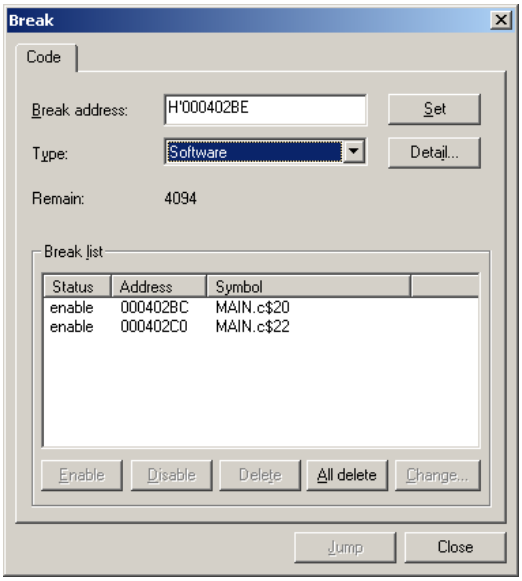
5.2 Position of Break point

When break points are set to some location and after that code is modified and build again, position of previously set break point will be decided as per following rule

- Break point will be set to the same source code line number
- If it is not possible to meet above condition, it will be set to the same address location

For example,

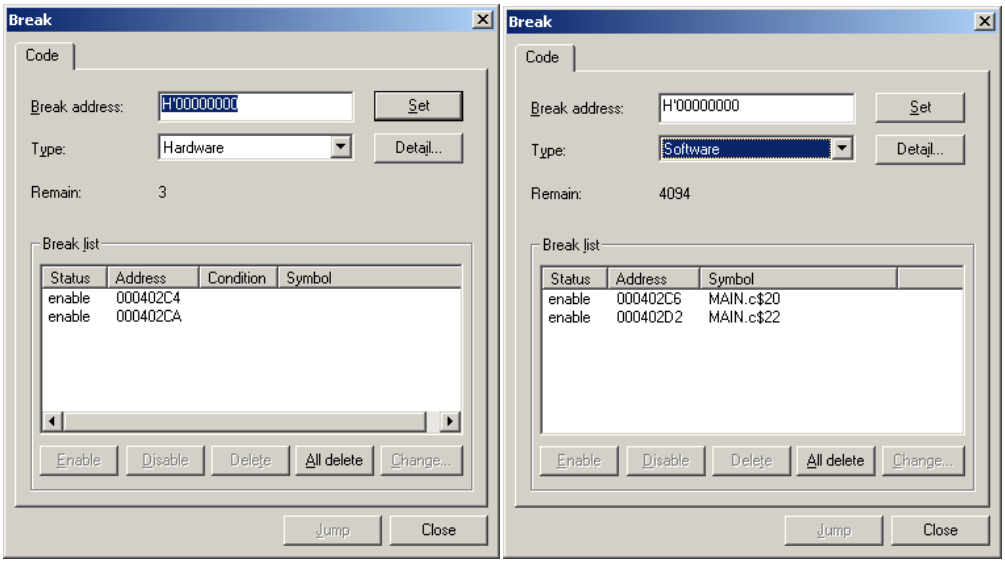


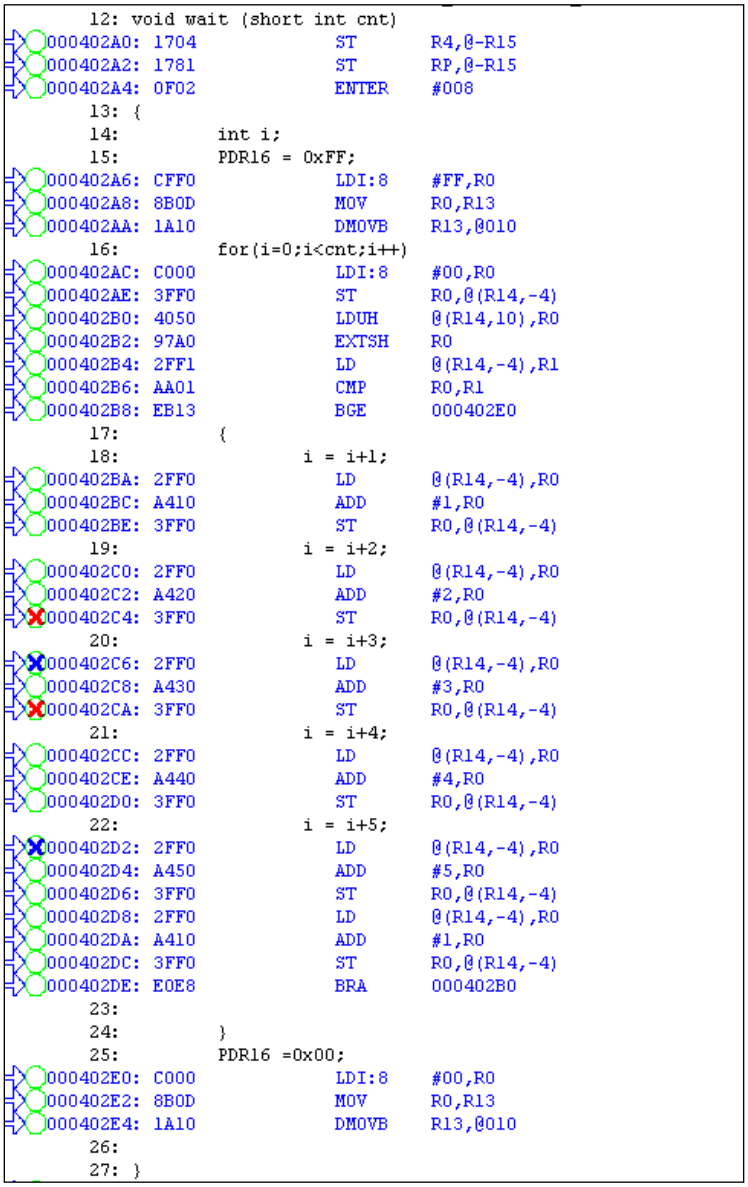


```
12: void wait (short int cnt)
13: {
14:         int i;
15:         PDR16 = 0xFF;
16:         for(i=0;i<cnt;i++)
17:         {
18:
19:                 __asm(" NOP");
20:                 __asm(" NOP");
21:                 __asm(" NOP");
22:                 __asm(" NOP");
23:                 __asm(" NOP");
24:                 __asm(" NOP");
25:                 __asm(" NOP");
26:                 __asm(" NOP");
27:                 __asm(" NOP");
28:                 __asm(" NOP");
29:
30:
31:         }
32:         PDR16 =0x00;
33:
34: }
35:
```

Let us change the code, replace couple of `__asm( "NOP");` instruction with some other instruction in *wait* function.

And let us assume that Breakpoint position changes as shown below





For Software breakpoint 1, this was at source code line no. 20, since after build there is valid instruction at that line; breakpoint is maintained at same source code line main.c$20

For Software breakpoint 2, this was at source code line no. 22, since after build there is valid instruction at that line; breakpoint is maintained at same source code line main.c$22

For Hardware breakpoint 3, this was at source code line no. 24, since after build there is no valid instruction at that line; breakpoint is maintained at same memory location H’ 000402C4

For Hardware breakpoint 4, this was at source code line no. 27, since after build there is no valid instruction at that line; breakpoint is maintained at same memory location H’ 000402CA

6. Events

6.1 How to Set Code and Data Events

Two types of Event can be set. Those are Code event and Data event.

Code Event:

Two code events can be set.

Code event can be set with *pass count* from 1 to 65535 and *address mask*.

Pass count: It is the count of times PC needs to pass the set *Address* before program execution stops at the *address*.

Address Mask: A pattern of characters, 32 bits long, used to select some of the bits from the *address*. For each bit set to one in *Address mask* field, corresponding bit from *Address* is compared for exact match with the corresponding bit hold by PC and for each bit set to zero, corresponding bit from *Address* is ignored. Detail explanation with example is given in coming chapters.

Data Event:

When Data event is set, program execution stops when the data at the *address* is accessed.

As explained in code event it is possible to set data event point with *Address mask*

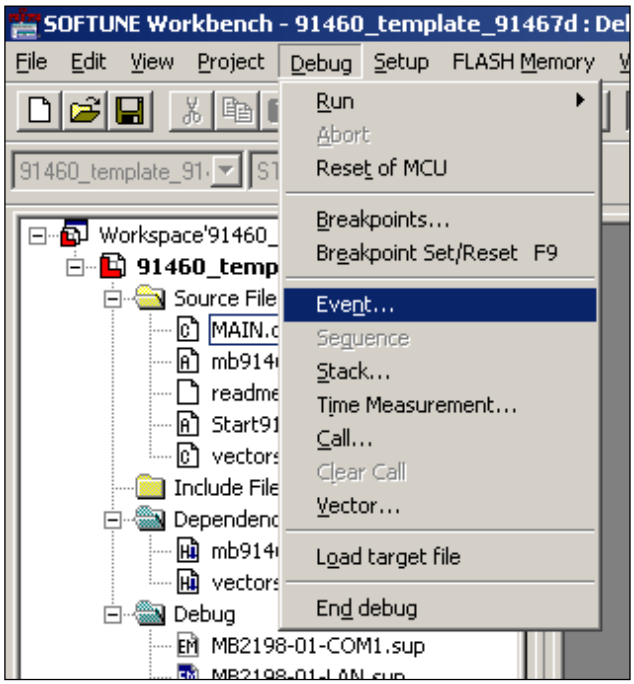
It is also possible to specify the *Data* content with *Data mask*, which is when read from or written to the *Address*, program execution will halt.

Further data can be specified with *size* option i.e. byte, half word and word.

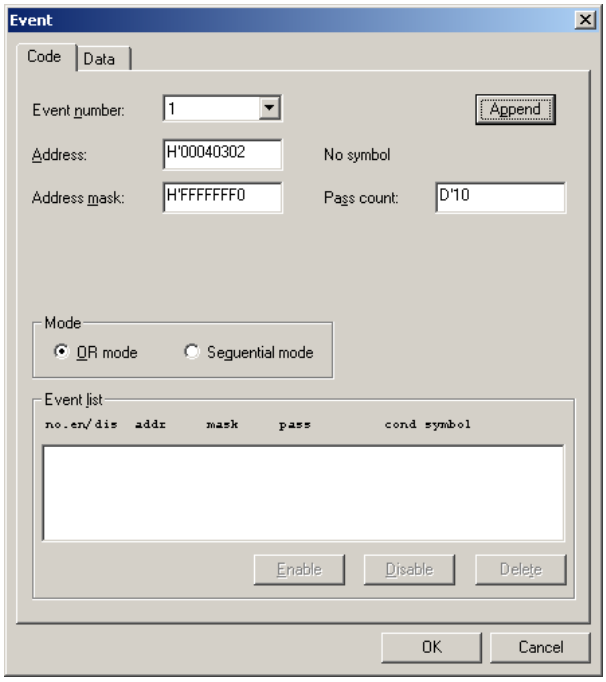
Two data event can be set.

6.1.1 Code Event

As shown below, *Event* dialog box is opened by clicking on *Event…* option on *Debug* pull down menu.

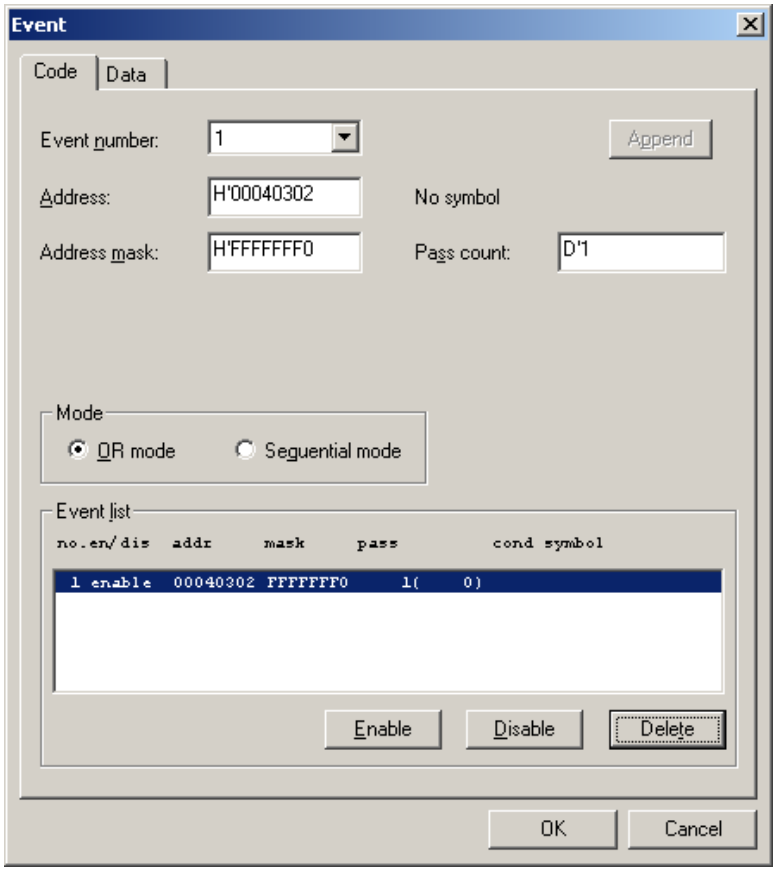


In the below dialog box, *Event Address* - Specifies the address at which the event occurrence condition is to be set. *Address Mask* specifies address mask. Under *Mode*, two possible modes can be selected. In *OR Mode,* events are triggered when the event 1 condition or the event 2 condition is established. In *Sequential mode* Events are triggered when the event 1 condition and event 2 condition are established in sequence.



When *address mask* is selected, for each bit set in *address mask* field, corresponding bit from *Address* is compared for the exact match with the corresponding bit of address hold by Program Counter i.e. Program will stop executing due to event break whenever following condition is met…

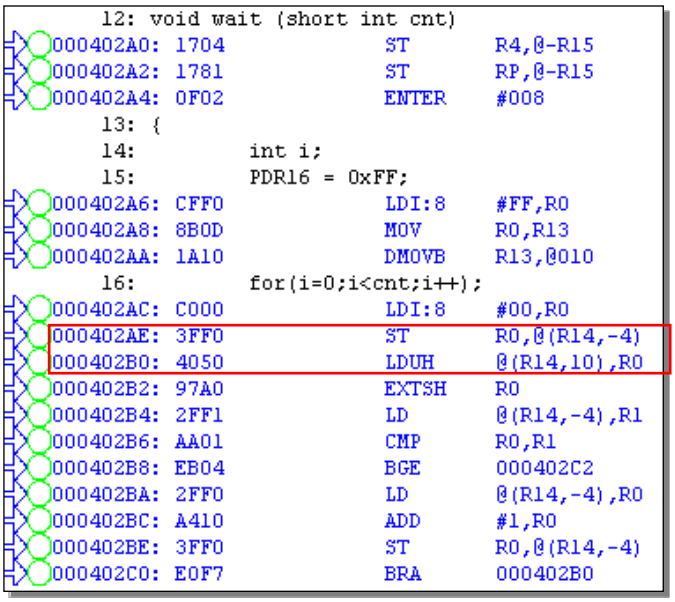
(Address hold by PC) & (Address Mask) = (address) & (Address Mask)



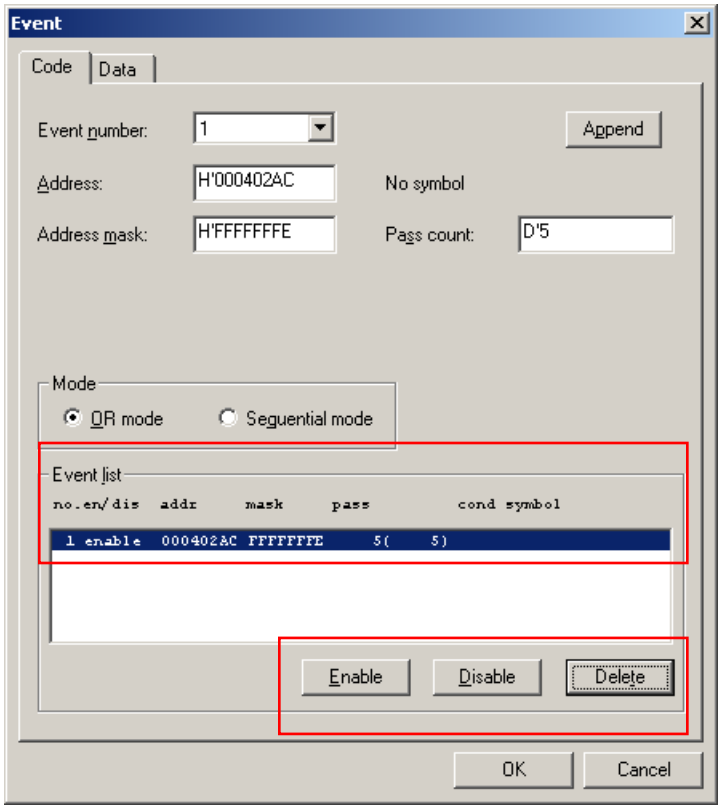
For example, if we set Code Breakpoint at address H' 00040302 and address Mask H'FFFFFFF0 and when program is executed it will stop at all the address (where there is valid instruction) from H' 00040300 to H' 0004030F

Similarly when we select *Pass count* (1 to 255), program will halt executing at *Address* when program counter executes instruction at the address selected in the *Address* field, as many numbers of times as that is selected by *Pass count*.

For example, if we set code break point at *Address* H'000402AC and select *Pass count* equal to D'5. When program is run, it halts due to hardware code break point when value of variable 'i' equal to D'5

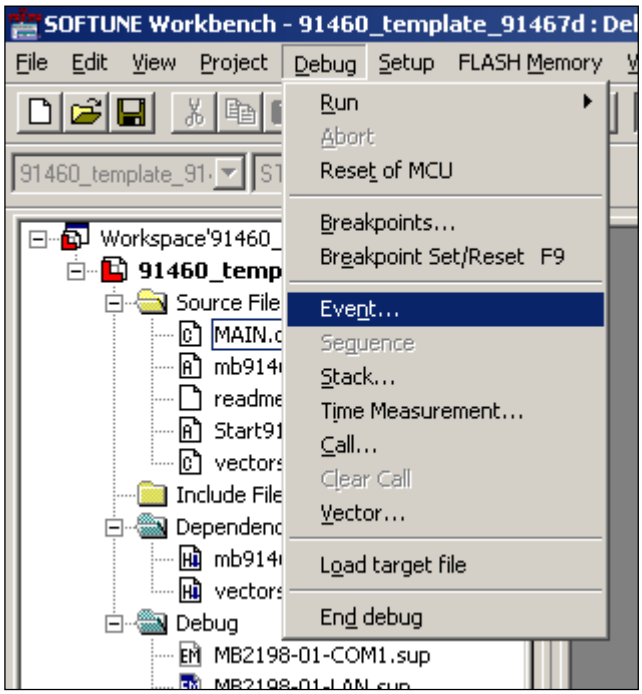


Event can be released by selecting it from *Event list,* and clicking on *Delete* button, alternately it can also be disabled by clicking *Disable* button

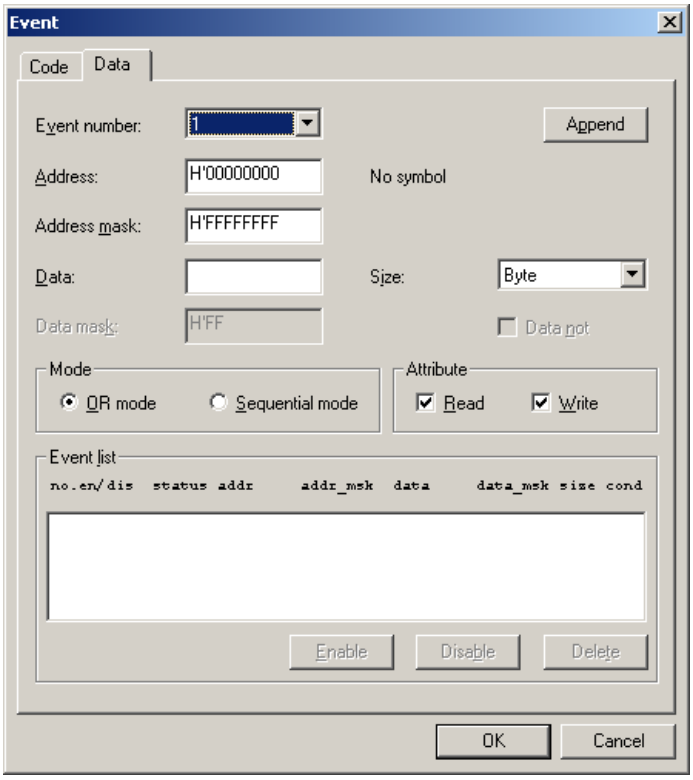


6.1.2 Data Event

As shown below, *Break* dialog box is opened by clicking on *breakpoints…* option on *Debug* pull down menu.

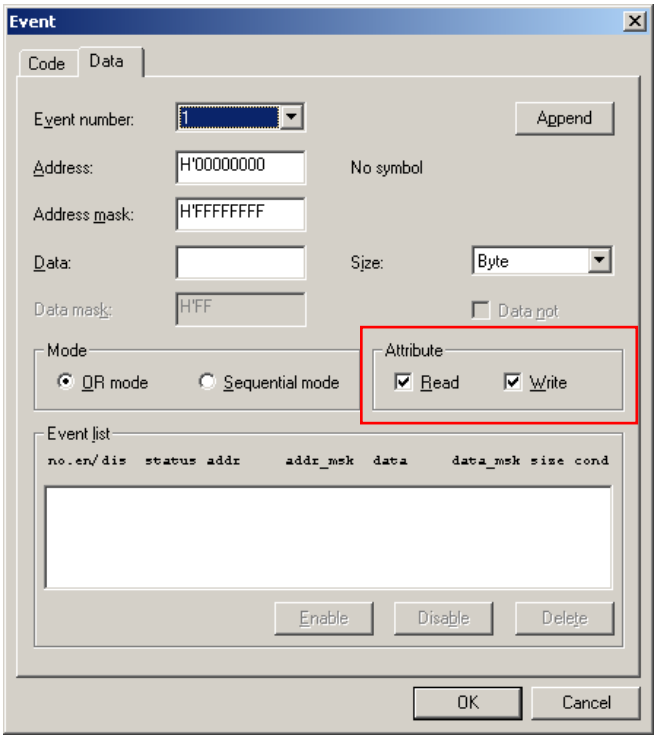


Click on *Data* tab,

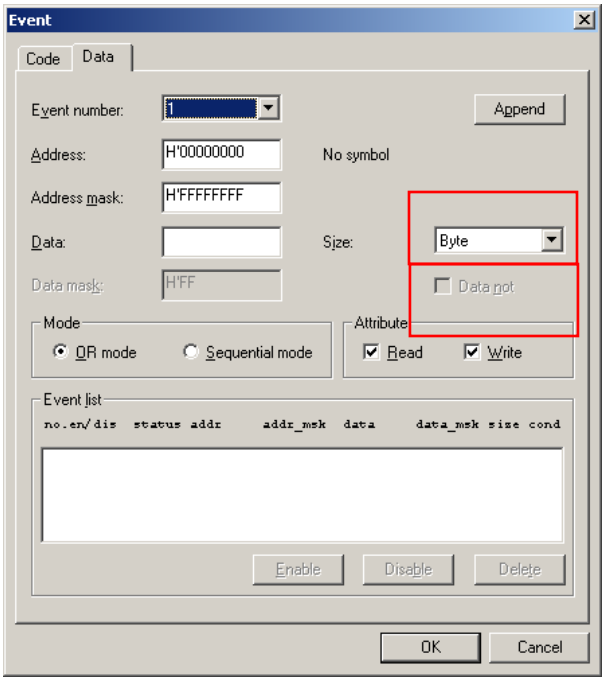


Address mask and Pass count feature work same as that of code event. *Data* specifies the data to be set as the event occurrence condition. *Data mask* specifies data mask. *Data not* specifies the condition when the data values do not match. *Size* specifies a data access size. *Attribute* specifies a data access attribute.

In *Event* dialog box, under *attribute* option it can be specified if *read* or *write* access at *Address* should cause a break.



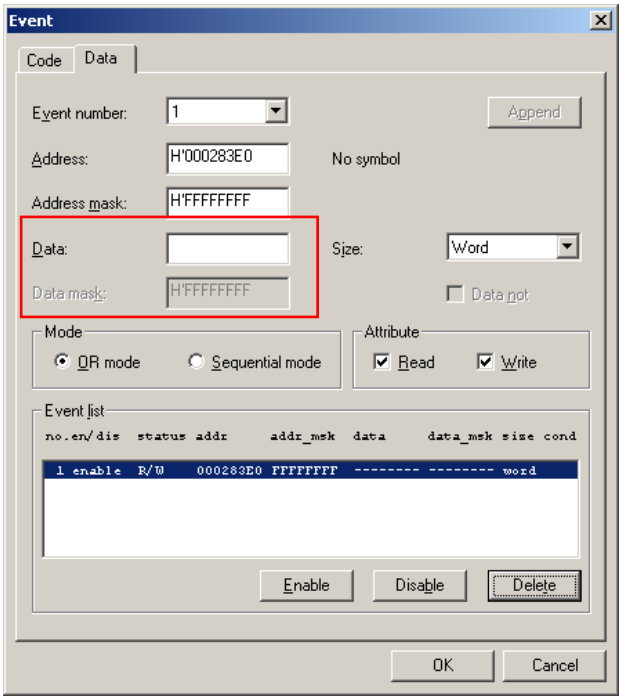
Under *size* option it is possible to select either *byte, halfword or word* type data.



Further, comparison condition can be selected among *Data equal to* or *Data not*.

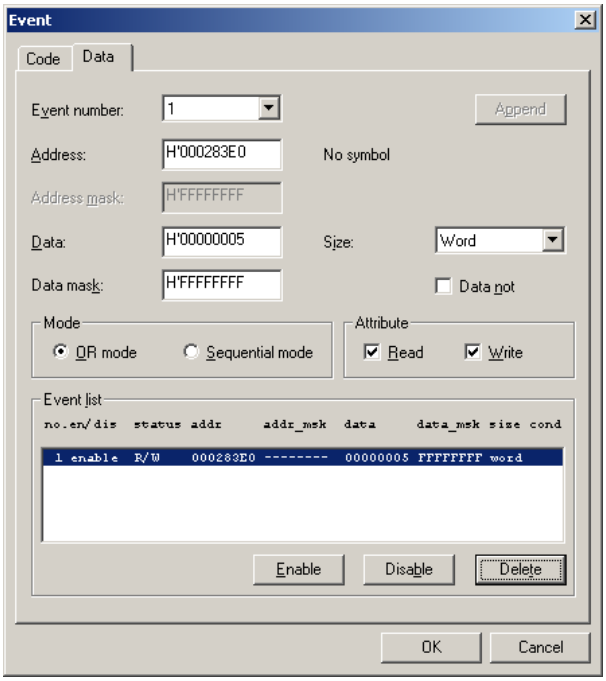
Note, that the Data and Data mask text boxes are only available if either *Data agreement* or *Data not* is selected.

When *Data* field is left empty, for all data access at *Address* for read or write access as selected by attribute, program execution will halt.



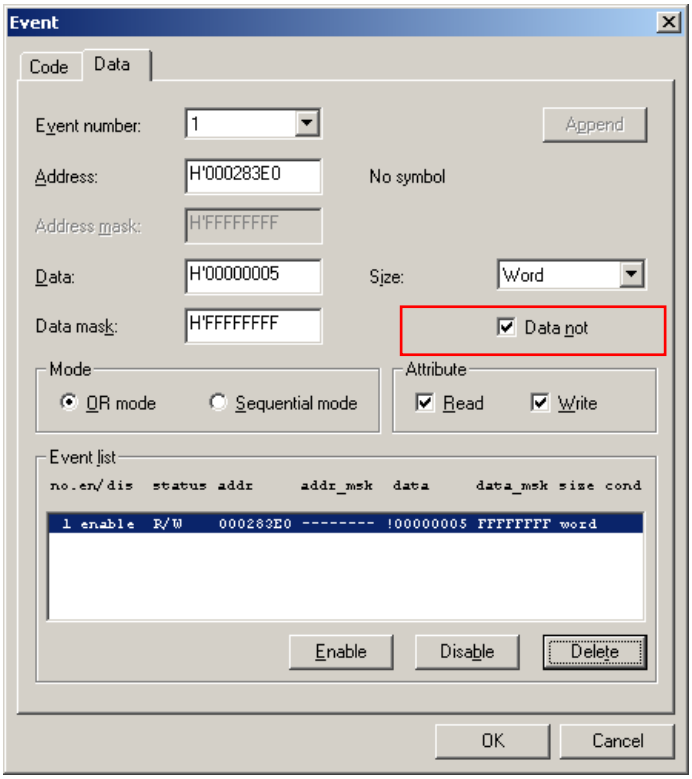
For example, if we set breakpoint as shown in above figure, program execution will halt every time when there is a read or write access to location H'000283E0

When *Data not* check box is unselected, for all data access at *Address,* the content *read* from or *written* to, is equal to that specified in *Data* field, program execution will halt.



For example, if we set breakpoint as shown in above figure, program execution will halt every time when data H' 05 is read from or written to location H'000283E0.

Similarly if we select *Data not* check box, for all data access at *Address,* the content *read* from or *written* to, is not equal to that specified in *Data* field, program execution will halt.

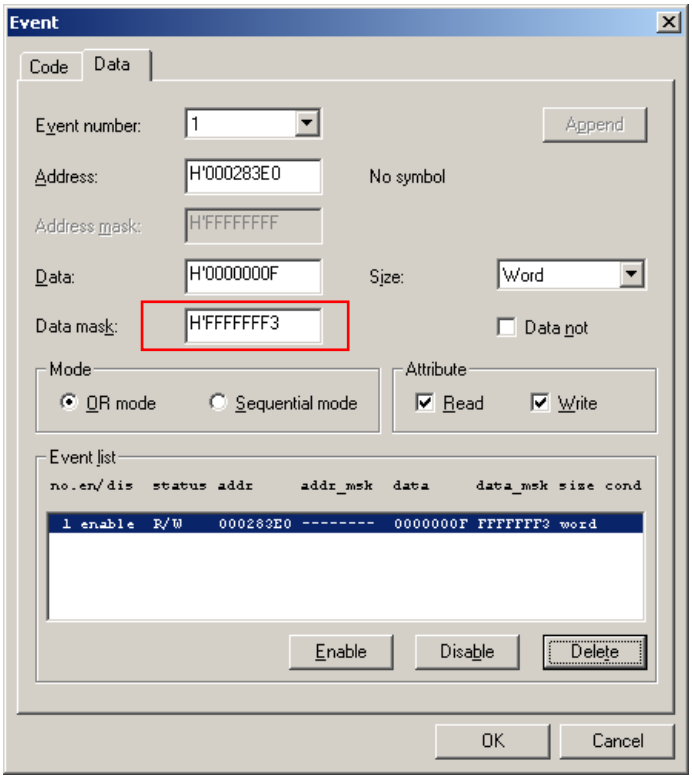


For example, if we set breakpoint as shown in above figure, program execution will halt every time when data other than H' 05 is read from or written to location H'000283E0.

When *Data mask* is selected, for each bit set in *Data mask* field, that bit from *Data* field is compared for the exact match with the corresponding bit of Data which is to be read or written at the break address by the MCU i.e. Program will stop executing due to break point whenever following condition is met…

(Data read by MCU at Break Address Selected) & (Data Mask) = (Data) & (Data Mask)

For example, if we set Data Breakpoint at address H' 000283E0, which is the address of a int type variable and if the attribute is set to read, comparison condition is set to Data agreement, data is set to H'000000F and Data Mask is set to H'FFFFFFF3.

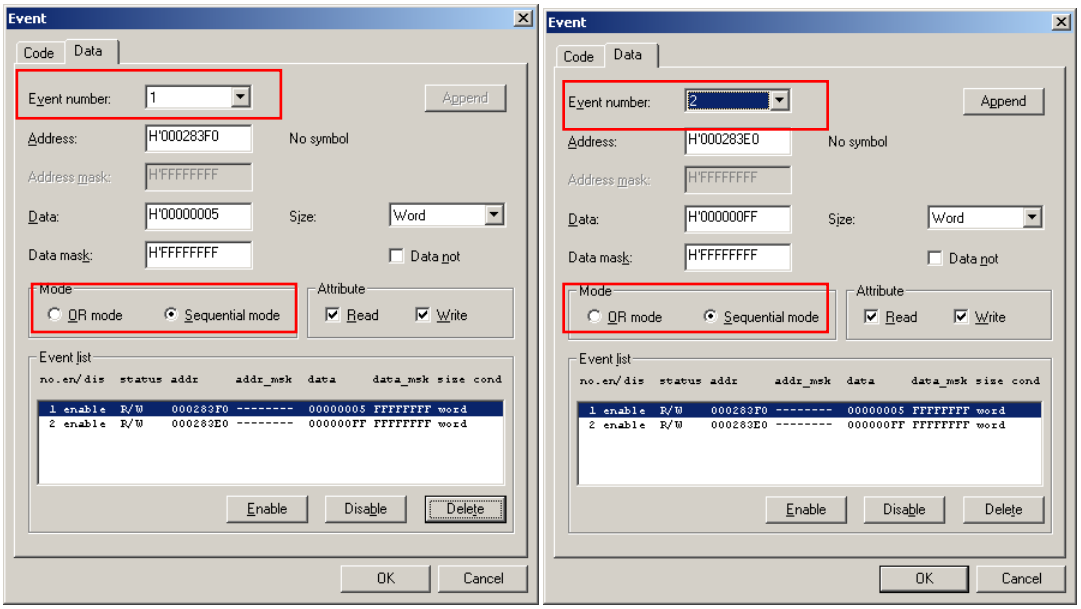


When program is executed it will stop when data, either H'00000C or H'000000D or H'000000E or H'000000F is read from break address H' 000283E0.

Now let us consider that we want in a following code, we want program to stop executing when `count = H'05` and `i = H'FF`. We can do this using sequential mode, while setting an event

```
void wait (short int cnt)
{
    int i;
    PDR16 = 0xFF;
    for(i=0;i<cnt;i++);
    PDR16 =0x00;

}

void main(void)
{
    int count = 0;

    __EI();                     /* enable interrupts */
    __set_il(31);               /* allow all levels */
    InitIrqLevels();            /* init interrupts */

    PORTEN = 0x3;               /* enable I/O Ports */
                                /* This feature is not supported by MB91V460A */
                                /* For all other devices the I/O Ports must be enabled*/
    PFR16 = 0x00;
    DDR16 = 0xFF;

    while(1)                    /* endless loop */
    {
       HWWD_CL = 0;
       count++;
       __asm(" NOP");
       wait (5000);
       __asm(" NOP");
    }
}
```

For this we need to set event 1 and 2 as described below, assuming *count* is located at H'000283F0 and *i* is located at H'000283E0



7. Trace

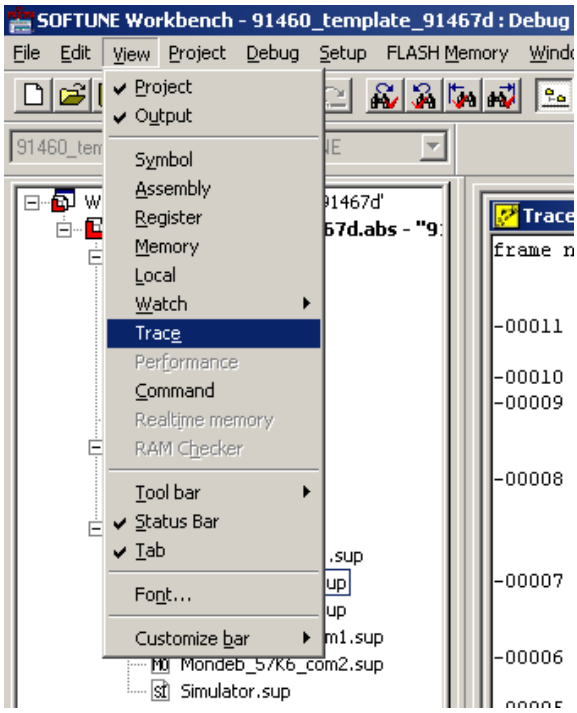
How To Use The Trace Buffer

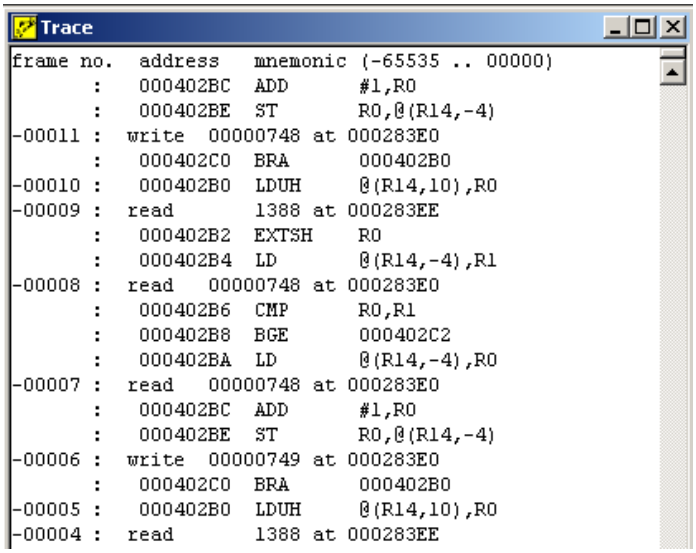
Trace function displays addresses and instructions executed so far. While execution of a program, the address, data and status information can be sampled and stored in the trace buffer. This function is called real-time trace.

In-depth analysis of a program execution history can be performed using the data recorded by real-time trace.

7.1 Trace Window

Trace window can be opened by clicking *Trace,* on *View* pull down menu.





In a Trace result display, the first column shows the frame number.

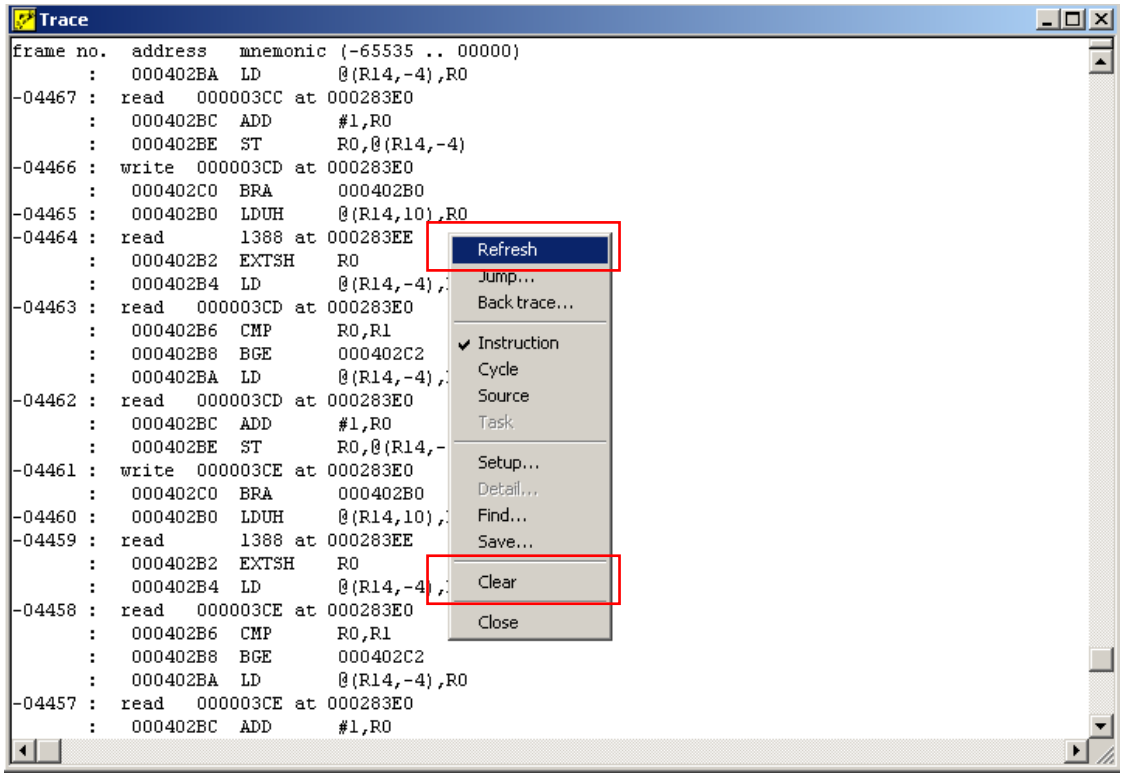
The second column shows the internal bus address.

The third column shows either the disassembled machine code (mnemonic) or a data transfer (inclusive data value).

Note that the last executed frame has the number 0 and all previous frames negative numbers.

Every time, after program run, to refresh a trace window right click on Trace Window, on a popup menu click on *Refresh*.

To clear previous data, right click on Trace Window, on a popup menu click on *Clear*.

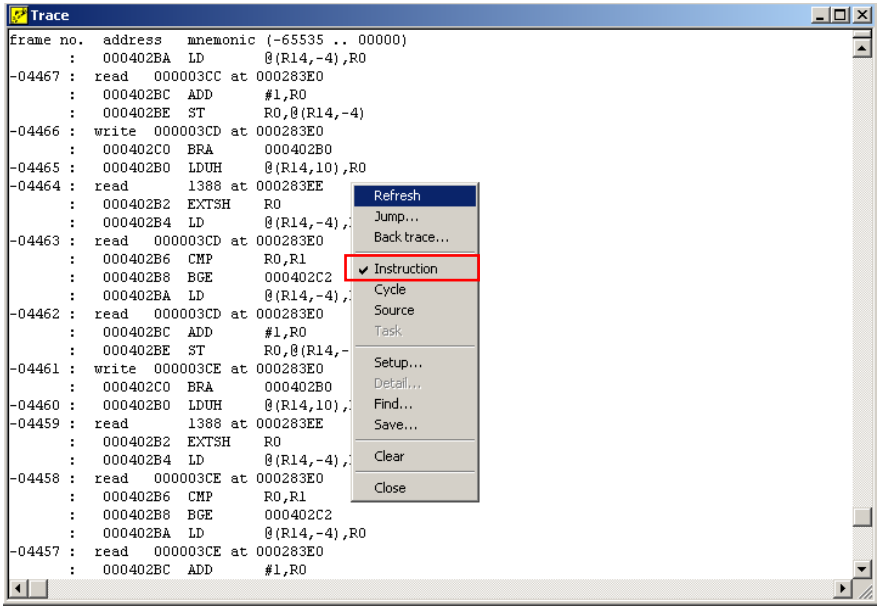


7.2 Trace View

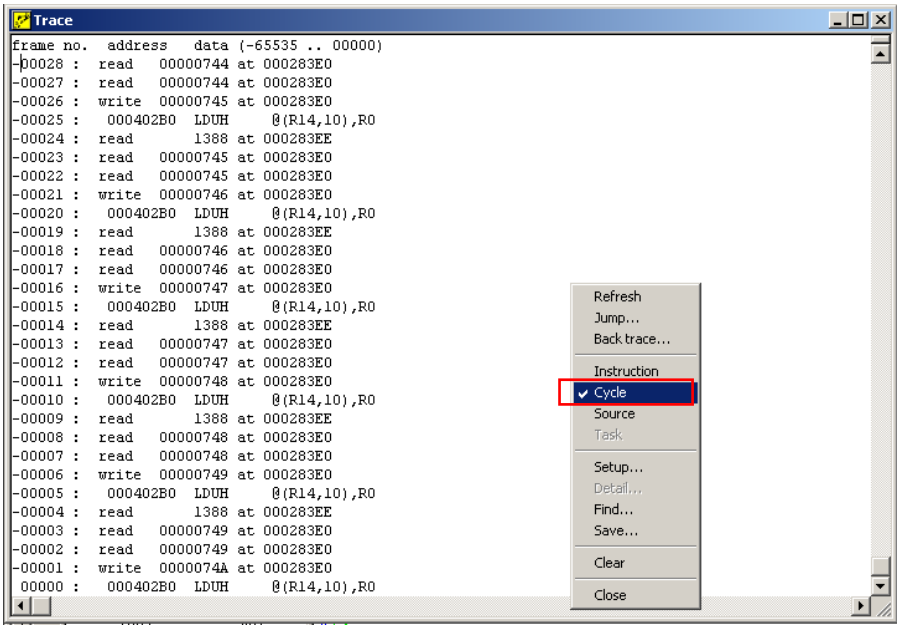
Trace view can be changed among Instruction view, raw data view or source view.

For Instruction view, right click on Trace window, on a popup menu click on *Instruction*.

Instruction view show instruction cycles using assembler mnemonics



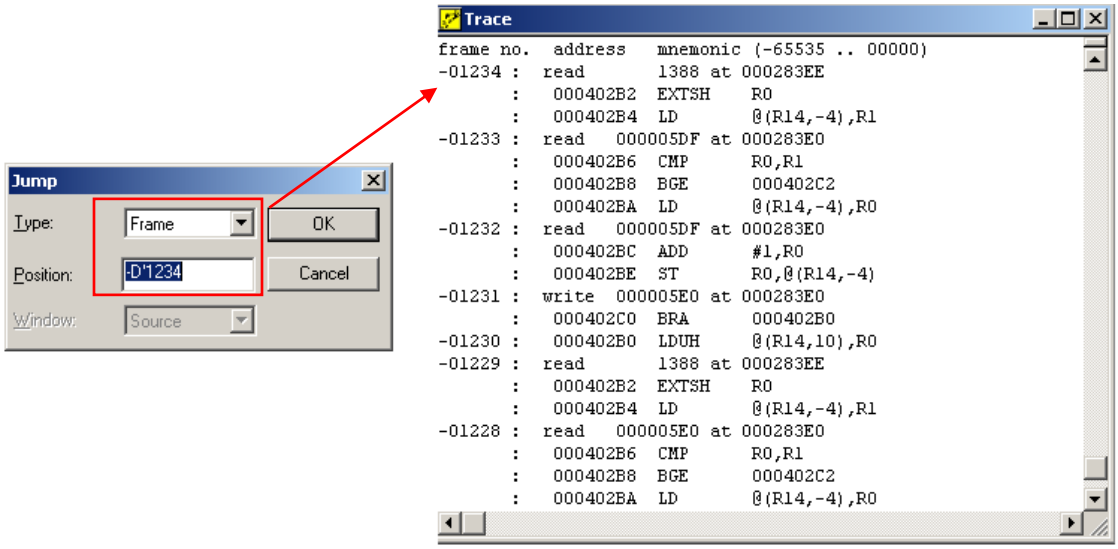
For raw data view, right click on Trace window, on a popup menu click on *Cycle*.



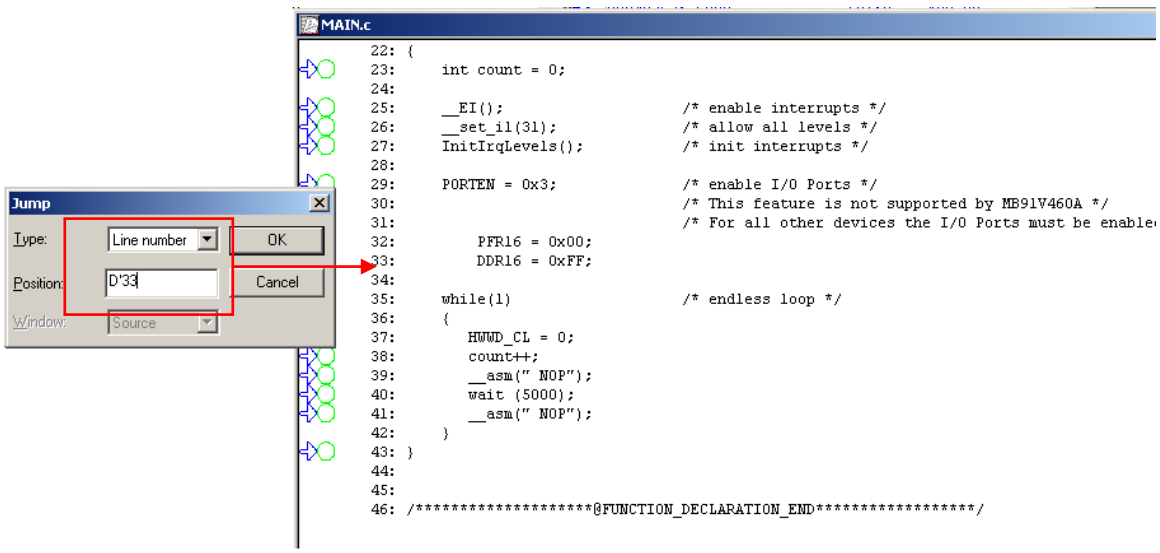
For Source view, right click on Trace window, on a popup menu click on *Source*. Source view shows program source code (e.g. C - language)

7.3 Trace Jump

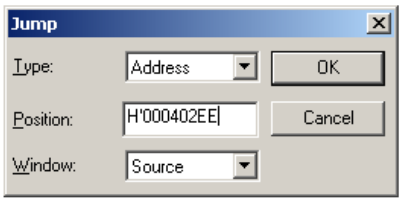
To move cursor to particular frame in trace window, right click on *trace* window, click on *Jump…,* on *Jump* dialog box, select *frame* under *Type* option and desired frame number under *Position* option and click *OK*

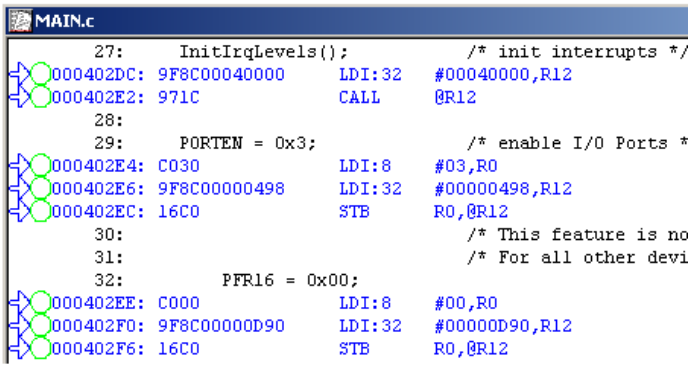


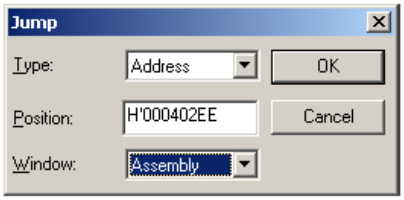
To view source code at particular line number, right click on trace window, click on *Jump…*, on *Jump* dialog box, select *Line number* under *Type* option and source line number under *Position* option, click *OK,* in editor window one can view source code at selected line number

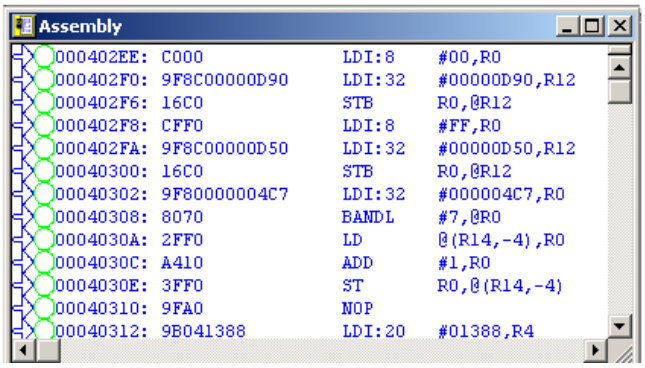


To view assembly instruction at particular memory location, right click on trace window, click on *Jump…*, on *Jump* dialog box select *Address* under *Type* option and memory location under *Position* option and than select either *Source* or *Assembly* under *Window* option.

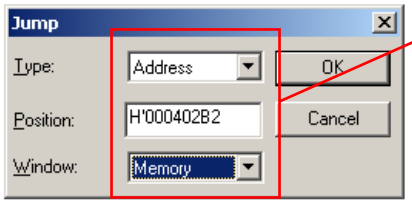


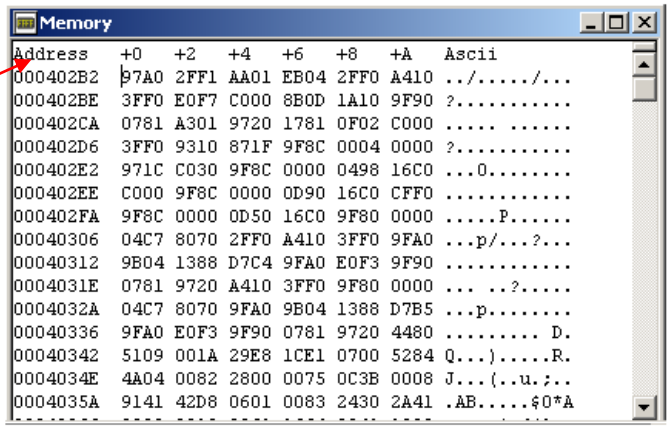






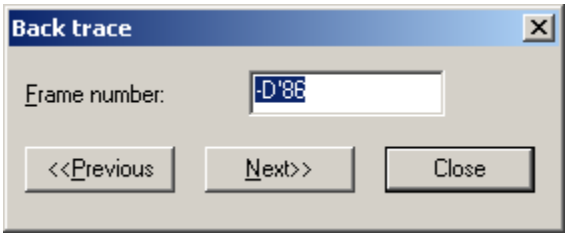
To view data at particular memory location, right click on trace window, click on *Jump…*, on *Jump* dialog box select *Address* under *Type* option and memory location under *Position* option and than select memory under *window* option.





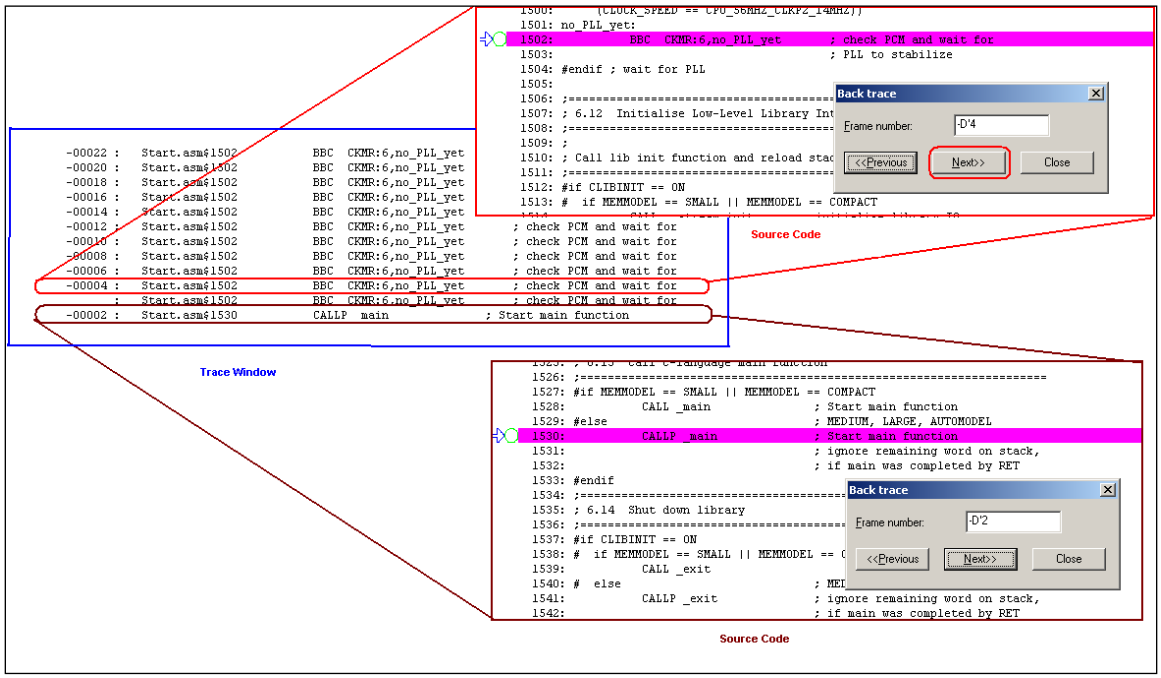
7.4 Back Trace

When selected *Source view,* one can use *Back trace…* functionality. Right click on editor window, click on *Back trace…, Back trace* dialog box will appear



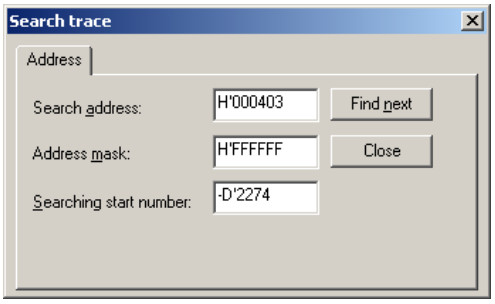
With this functionality, one can traverse back and forth in source window in a same sequence, in which source code is executed. Source code at particular frame number will be highlighted by pink background while traversing.

For example, in figure below, trace window shows address and instruction at frame number -D'2 and -D'4 executed in sequence. In a *back trace* dialog box, if we select *frame number* –D'4, corresponding line at D'1502 in source code will be highlighted, if we click next, source code line number D'1530 will be highlighted which corresponds to *frame number* –D'2.

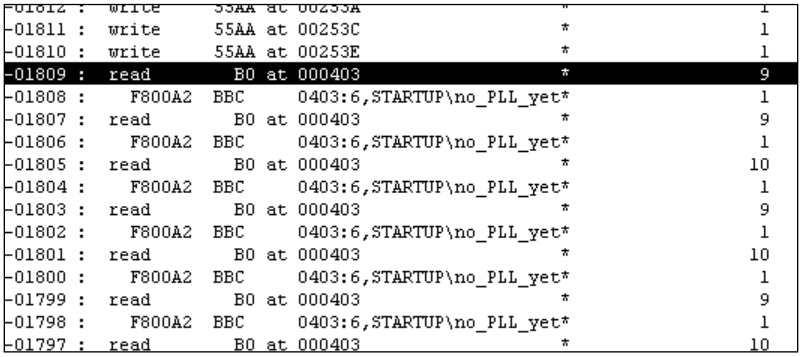


7.5 Search Trace

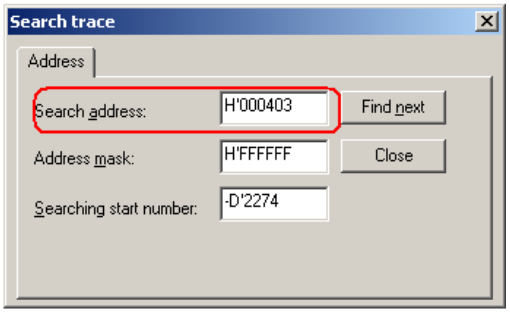
To find particular memory location in trace window, one can use *find* feature. Right click on Trace window, on a popup menu click on *Find. Search trace* dialog box will appear.



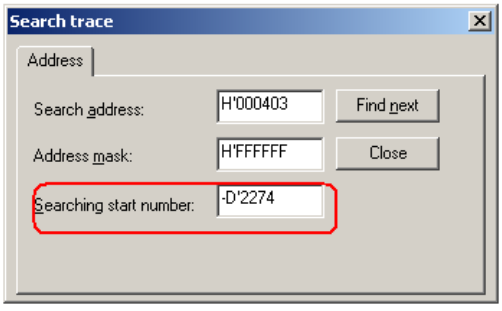
The line having information about searched memory location is highlighted with black background.



The required memory location is entered in *Search address,*



The frame number from where to start the trace search is entered in *Searching start number*

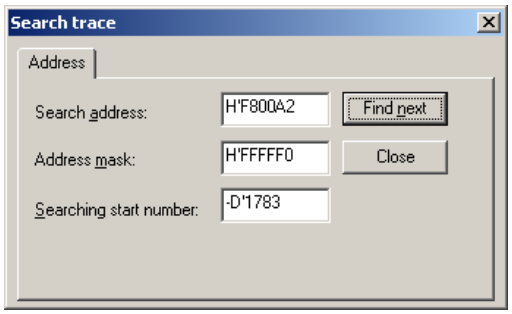


When *Address mask* is selected, for each bit set in *Address mask* field, that bit from *Search address* field is compared for the exact match with the corresponding bit of the memory location read by the debugger from trace data buffer. I.e. The line having the frame number for which following condition is met, will be highlighted in Trace window

(Memory location read by debugger) & (Address Mask) =

(Search address) & (Address Mask)

For the trace search, configured as shown in the figure below,

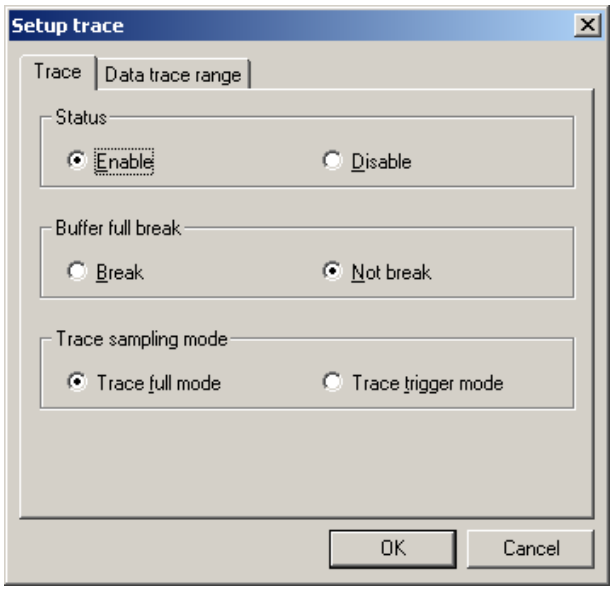


When every time *Find next* button is clicked, trace frame, containing memory address in the range of H'F800A0 to H'F800AF is highlighted. When search reaches the first frame number (-D'1) it rollbacks and search continues from last frame (-D'65535).

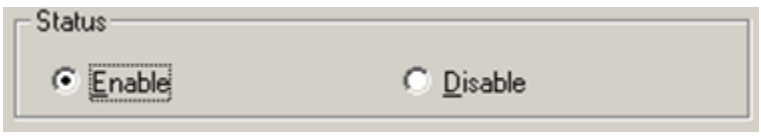
7.6 Trace Setup

Trace functionality can be configured by using *Setup Trace* dialog box.

Right click on *trace* window, on pop up menu, click on *Setup...*

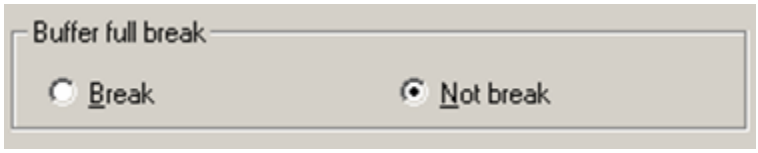


Trace data buffering can be enabled or disabled by selecting *Enable* or *Disable* radio button, under *Status* option.

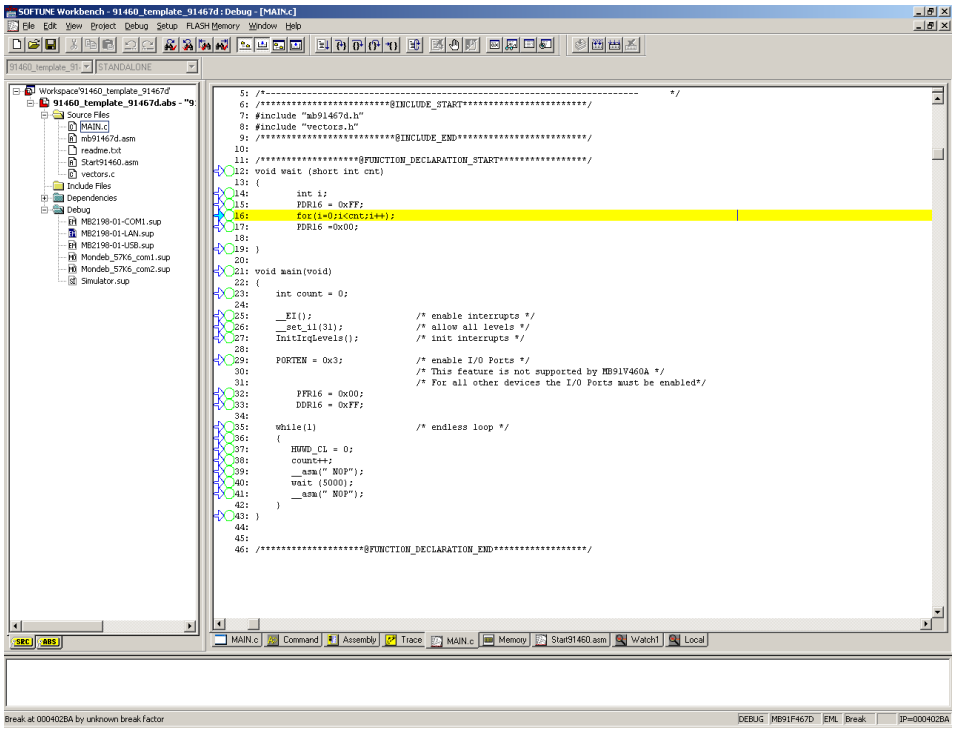


Trace data is buffered in the trace buffer. The trace buffer becomes full some time during debugging because its size is finite. When the trace buffer becomes full, the program being executed can be stopped. Trace buffer full break is set by selecting *Break* radio button under *Buffer full break* option.

MB2198-01 Emulator has 64K frames internal trace buffer and 256M frames External trace buffer.

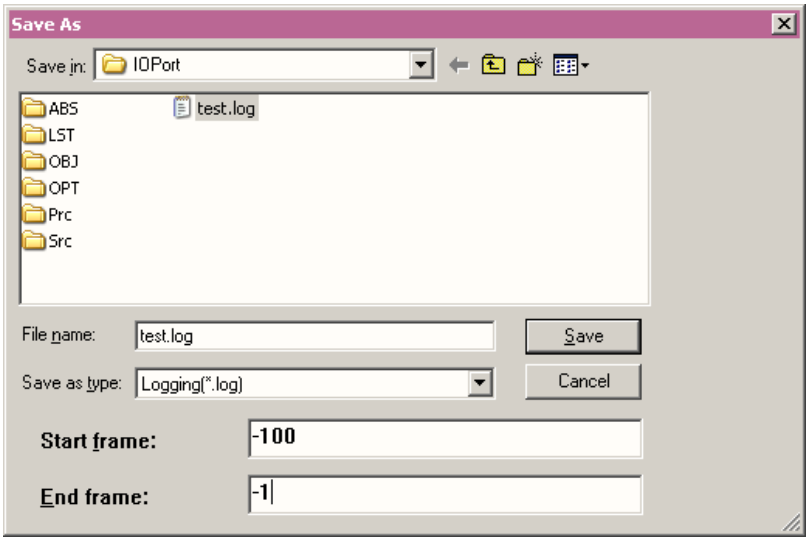


When program is halted by trace data buffer full break, corresponding line will be highlighted with yellow back ground.



7.7 Saving Trace Data

Since Softune version V30L34R05 it is possible to save the trace data beginning with a starting frame and ending with an end frame. Please enter negative numbers due to negative trace frames.



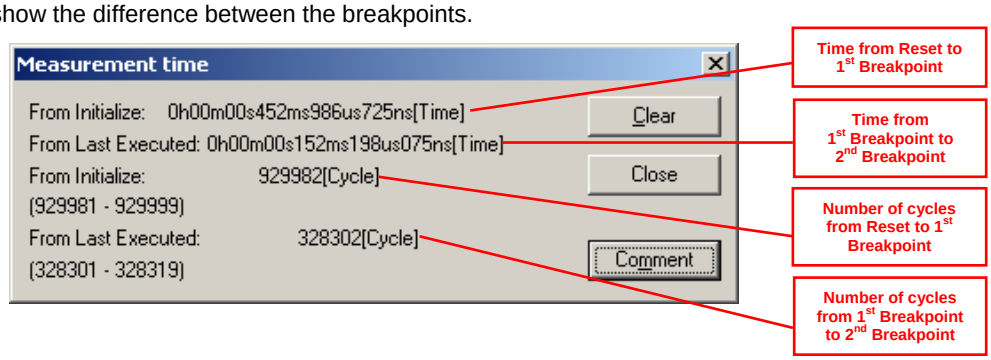
The save dialog is available by right-clicking in the trace window and choosing *Save file … .*

8. Time Measurement

How to Use Time Measurement Function

The Time Measurement is the simplest method of measurement. A 64-bit counter is used for this measurement resulting maximum cycle count of 18446744073709551615.

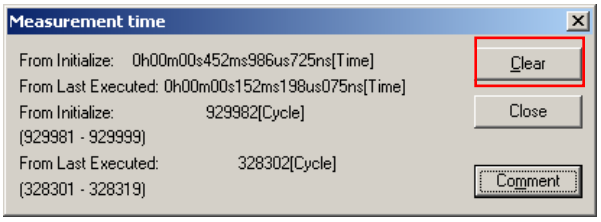
In this method two breakpoints have to be set. After the program has run from breakpoint 1 to 2 the Time Measurement will show the difference between the breakpoints.



Now, take a look at example as shown below, in a *main()* function, two breakpoints are set. In *a wait()* function *PORT00* is set to high when entered in function and then set to low when left the function.

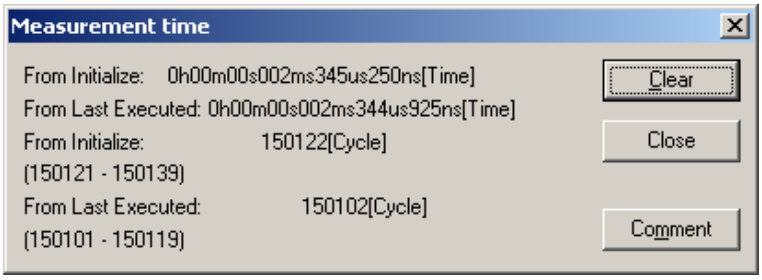
```
23: void main(void)
24: {
25:
26:         DDR00 = 0xFF;
27:         __asm(" NOP");
28:         wait (5000);
29:         __asm(" NOP");
30:
31: }
32:
33:
34:
35: void wait (int cnt)
36: {
37:         int i   ;
38:         PDR00 = 0xFF;
39:         for (i = 0; i< cnt ; i++);
40:         PDR00 = 0x00;
41:
42:
43: }
```

On Debug pull down menu click on *Time Measurement…*, Click on Clear if there are any time entries not equal to zero, Click on Close then.



Run the program. The MCU will stop at the beginning of the main function (1st breakpoint). This first stop initializes the time measurement counter. Next, run the program till the second breakpoint.

Now on Debug pull down menu click on *Time Measurement…*, one can find the following information

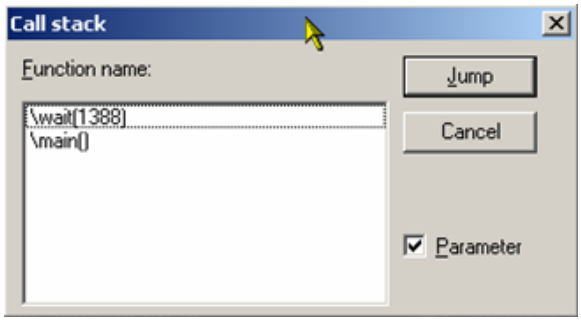


9. Call Stack

How to Use Call Stack Feature

Usually, a program is a set of several subroutines. For this reason, as debugging advances, function calls of several stages occur. For example, one routine calls another and the called routine further calls another.

Call stack window can be opened by clicking on *Stack* under *Debug* menu



The call stack retains the relationship between function calls. Clicking a function name from the function name list immediately displays information for the function in the Source Window.

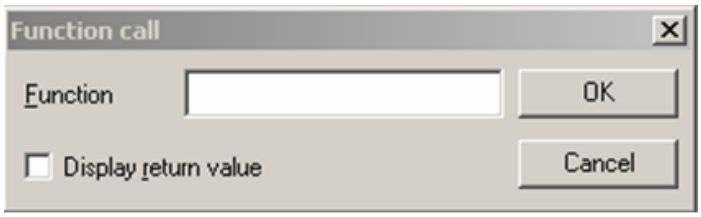
The function written in the lowermost line of the function name list is the main function. This main function calls the function above it. The called function further calls a function above it. In this way, the function written in the uppermost line is the function in which the current PC exists. When return is executed, functions are deleted in turn from the function name list, starting from the uppermost line.

When a check mark is set to *Parameter*, an argument value is displayed after each function name, as shown in above figure. When no check mark is set, only parentheses "( )" are displayed after each function name.

10. Function Call

How to Use Function Call Feature

The specified function can be started during debugging without reference to the flow of the program. This function is known as function call. This feature can be used when execution is stopped. After MCU reset one should at least execute one instruction before using this feature. The *Function call* dialog box is opened by right clicking on *Call…* on Debug pull down menu



When the function call dialog box opens, specify the function you want to call with a correct argument. If a breakpoint is set in the called function, the program stops at this breakpoint. When processing of the called function is terminated and control is returned, the function call result dialog box opens, if *Display return value check box* is checked. The PC then returns to the value before the function was called.

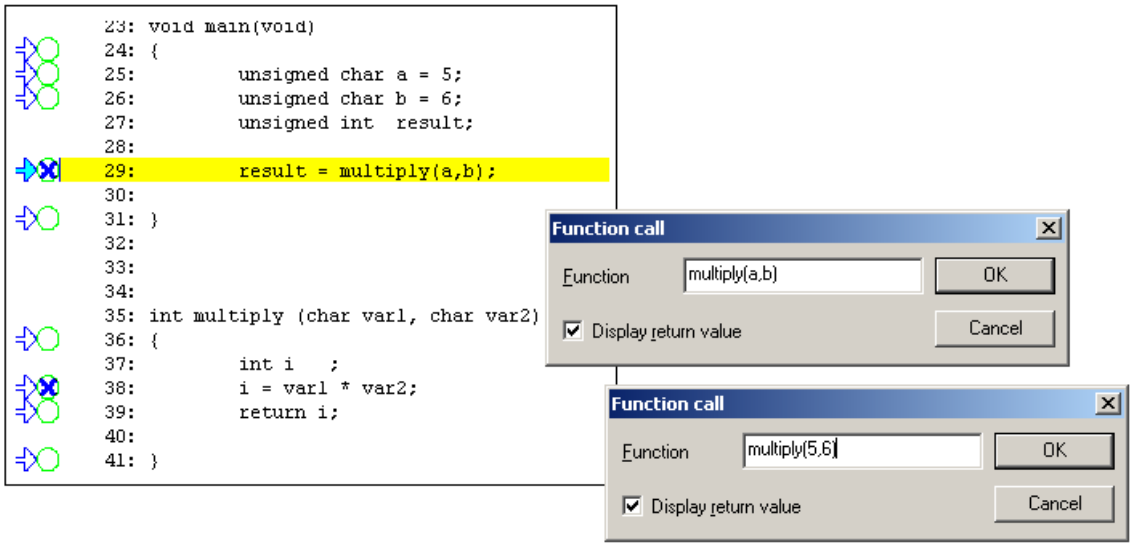
To understand the function, let us consider following example code

```
23: void main(void)
24: {
25:         unsigned char a = 5;
26:         unsigned char b = 6;
27:         unsigned int  result;
28:
29:         result = multiply(a,b);
30:
31: }
32:
33:
34:
35: int multiply (char var1, char var2)
36: {
37:         int i   ;
38:         i = var1 * var2;
39:         return i;
40:
41: }
```

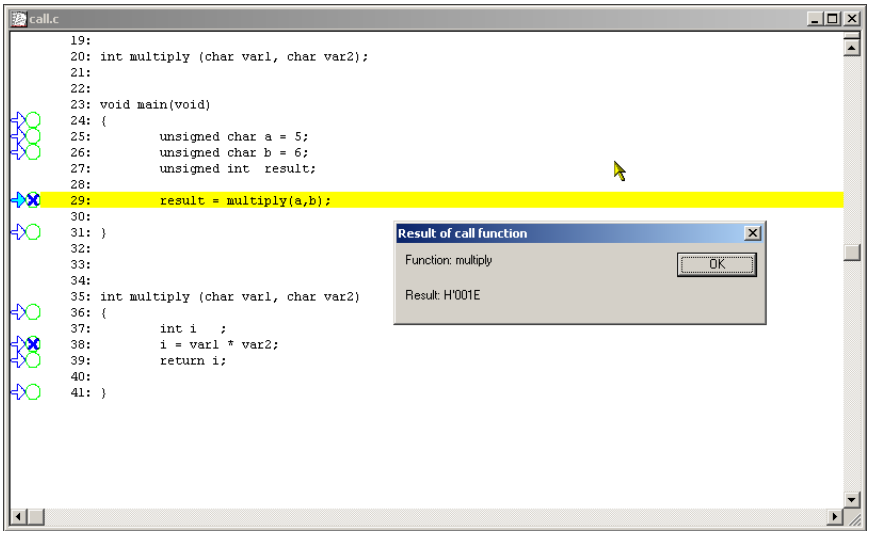
Let us set break point at line no. 29 and line no. 38. Start executing program, at first it halts at first break point. Now open *Function Call* dialog box (Debug -> Call…)

When the function definition is, *int multiply (char var1, char var2),* specify the function call as follow

- `multiply (5, 6);` Where a constant value is directly specified
- `multiply (a, b);` Where variable ‘a’ & ‘b’ is directly specified



Since break point is set in the called function, the program stops at this breakpoint. When run again at the end of processing of called function, control is returned, the *Result of call function* dialog box as shown below opens. The PC then returns to the value before the function was called.



11. Vector

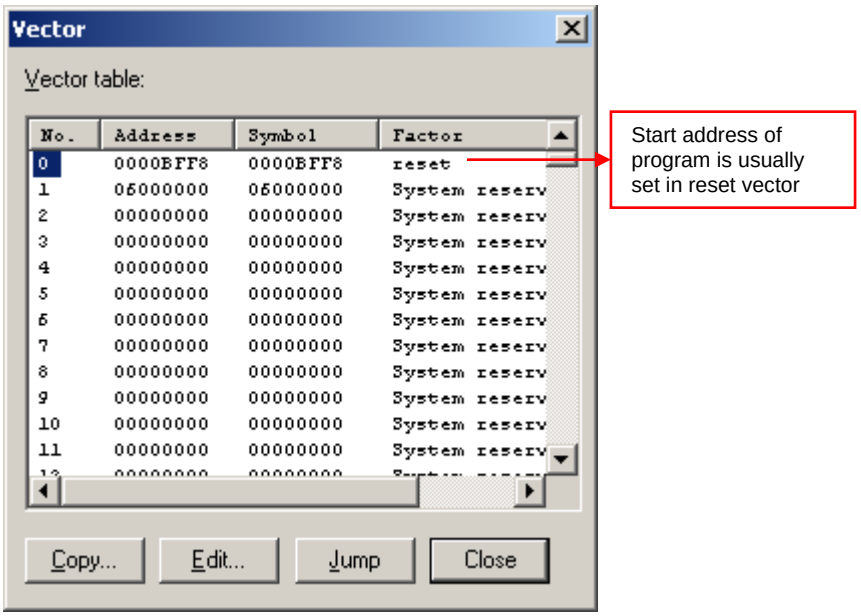
How to Use Vector Feature

An interrupt vector is the memory address of an interrupt handler, or an index into an array called an interrupt vector table or dispatch table. Interrupt vector tables contain the memory addresses of interrupt handlers. When an interrupt is generated, the processor saves its execution state via a context switch, and begins execution of the interrupt handler at the interrupt vector.

11.1 Display and setting vectors

11.1.1 Display

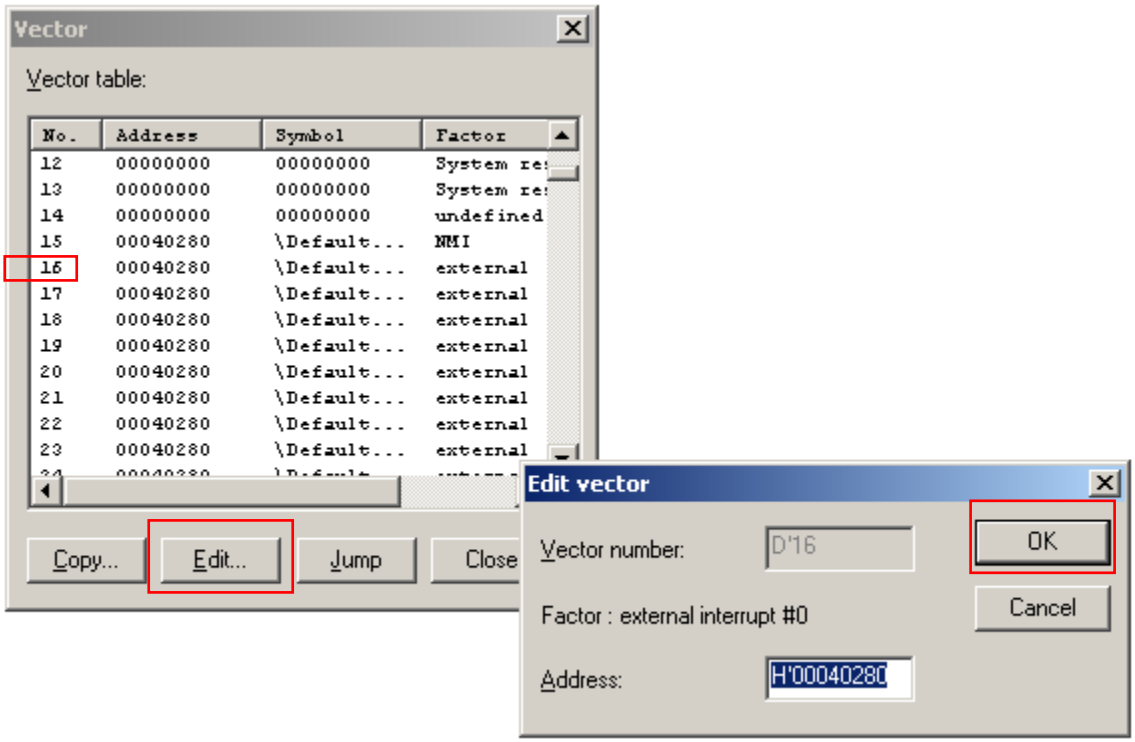
To display Interrupt vector table, click on *Vector…* at *Debug* pull down menu



11.1.2 Setting an address

Change the address set in a vector in the following procedure:

Select a vector table number and then click the *Edit* button. The vector edit dialog box shown below opens.



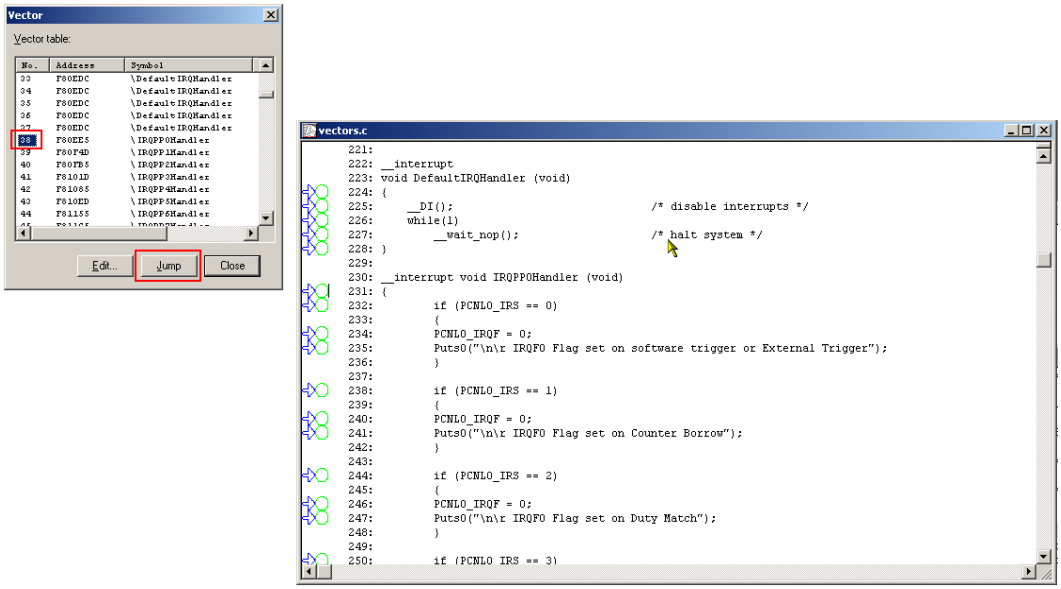
Set an address and then click the [OK] button.

11.2 Jump

Display the source of the program stored at the address set in the vector table in the following procedure:

1. Select a vector number.

Click the [Jump] button.

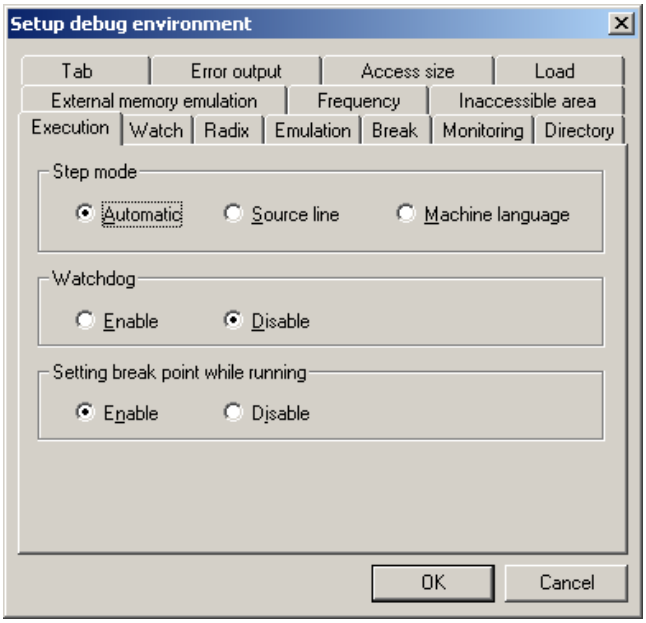


If the starting address of the program set in the vector table is incorrect, the source cannot be displayed (disassemble display).

The jump function merely displays the jump destination program; it does not update the program counter to move control to the address set in the vector table.

12. Debug Environment Setup Procedure

12.1 Execution



Step mode

- Automatic

  Automatically sets the step unit according to the window display state. If it is a source code view, step will pass through each source line no. If it is a Mix mode code view, step will pass through disassemble source line no.

- Source Line

  Executes the step in units of source lines.

- Machine Language

  Executes the step in units of machine languages.

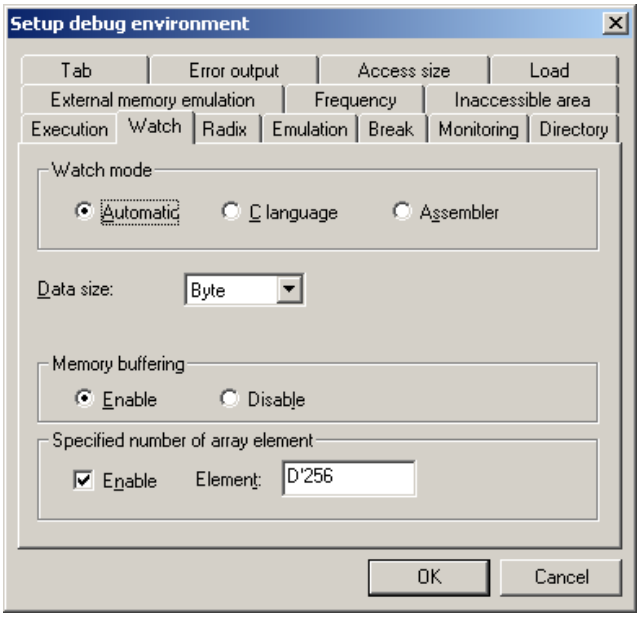
Watchdog

Specify whether to enable or disable the watchdog timer at program execution.

Setting break point while running

If "Setting break point while running" is enabled, it is also possible to break settings even when executing a user program.

12.2 Watch



Watch Mode

- Automatic

  Sets the watch mode automatically according to the analysis result. Refer 4.4 & 4.3

- C Language

  Sets the C language mode (interpretation as C language expressions).

  Refer 4.4 & 4.3

- Assembler

  Sets the assembler mode (interpretation as assembler expressions).

  Refer 4.4 & 4.3

Data Size

Sets the display size in the assembler mode to either byte, word, long, single or double

Memory Buffering

- Enable

  In case of variables as arrays or structures, memory of whole variables is read.

  They are accessed by size of the top variable.

- Disable

  In case of variables as arrays or structures, the memory of each variable is read.

Specified number of array element

- Enable

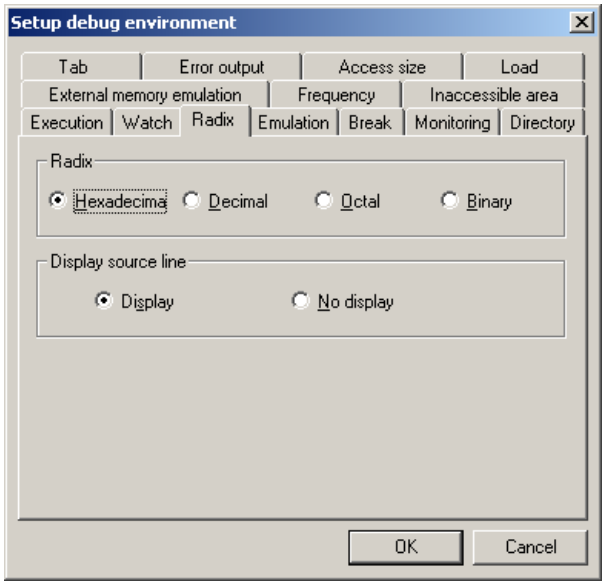
  Debugger displays a warning dialogue in case of bigger array element than the number of array-element that you limited, when you register or expand an array with a watch variable.

- element

  One can specify number (a default is D’256) of array element.

  The default of this control is "Enable".

12.3 Radix



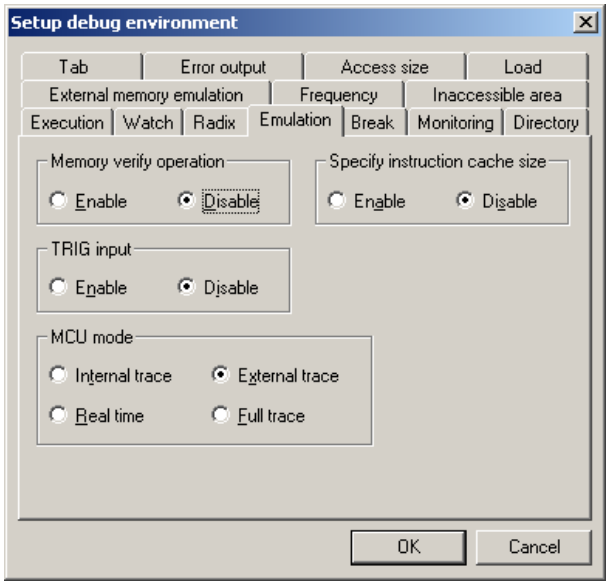
Radix

Sets the default base number for numerical value.

Display Source Line

Switches source line display to source line non display or vice versa for Trace- Instruction view.

12.4 Emulation



Memory Verify Operation

Specifies whether to verify memory when data is written to memory.

TRG Input

Specifies whether to enable or disable TRIG pin input. For more information please refer MB2198_01 Hardware manual (MB2198-01-CM71-00413-2E).

MCU Mode

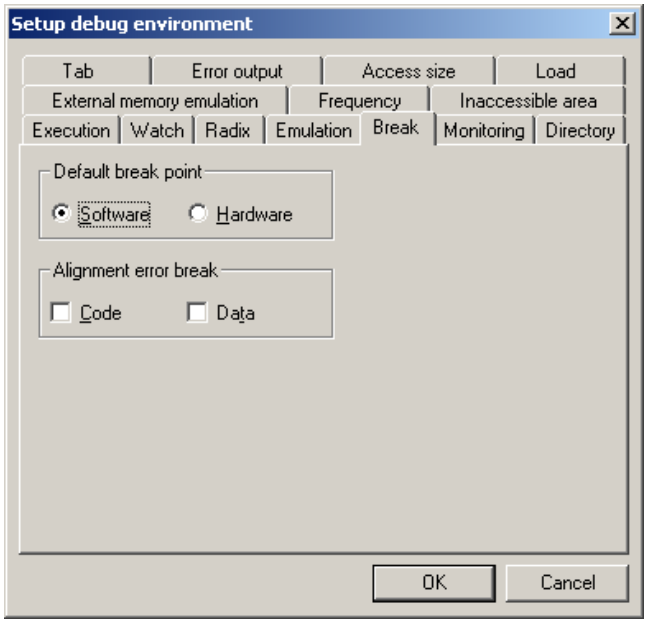
Specifies an MCU operation mode (internal trace, external trace, full trace mode or real-time mode).

Specify instruction cache size

Sets whether to automatically flush instruction cache.

12.5 Breakpoint

During program execution a break point can be set if the following set up is done:



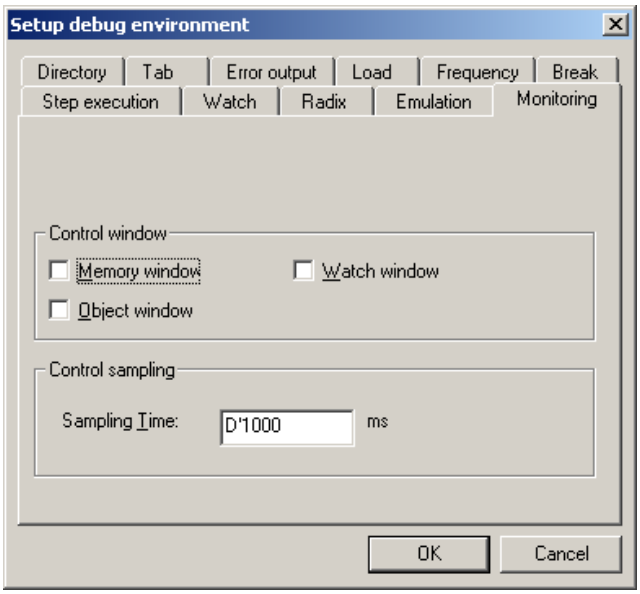
Default break point

Specifies the default type of the code breakpoint.

Alignment error break

Specifies whether to suspend MCU execution when an alignment error occurs

12.6 Monitoring



Control Window

- Memory Window

  Specifies whether to monitor the Memory Window.

- Watch Window

  Specifies whether to monitor the Watch Window.

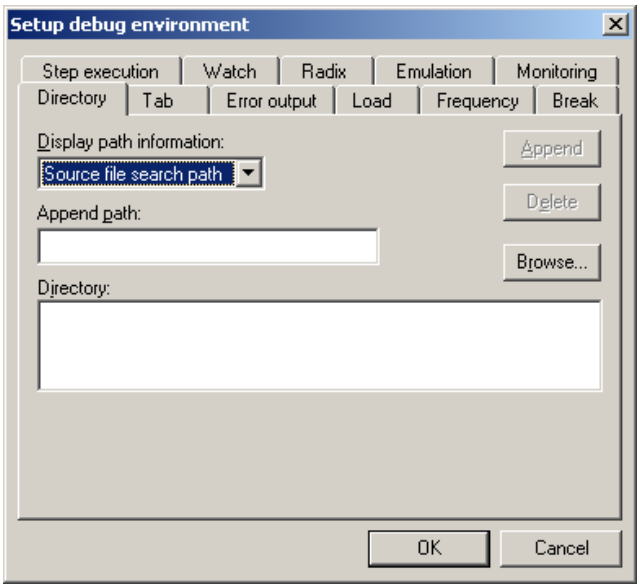
- Object Window

  Specifies whether to monitor the Object Window.

- Control Sampling

  Specifies sampling time for Watch window, Memory window and Object window

12.7 Directory



Display path information

Specifies the path information to be displayed.

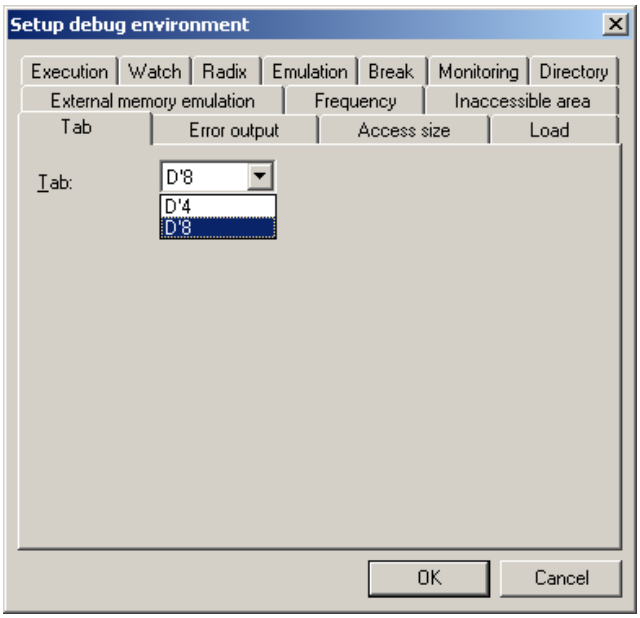
Append path

Sets the path to be added.

Directory

Displays the currently set directory.

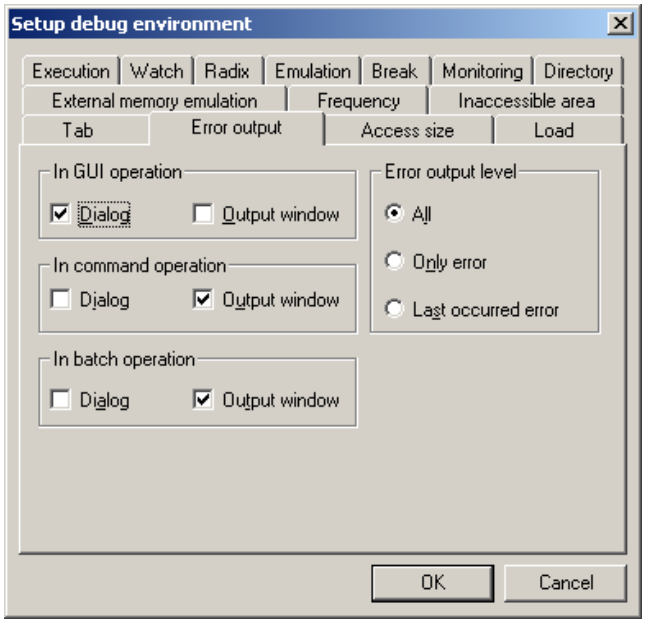
12.8 Tab



Tab

Specifies the Tab. (D'4/D'8)

12.9 Error output



In GUI Operation

Specifies where to output an error at GUI operation.

In Command Operation

Specifies where to output an error at command operation.

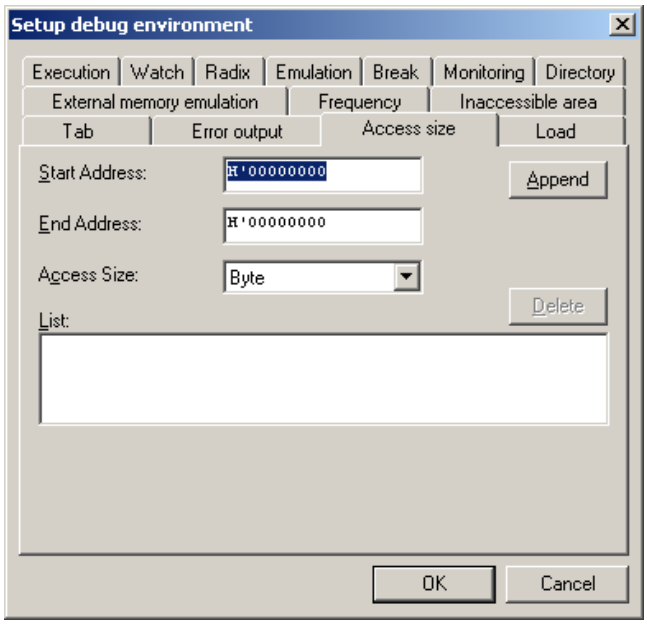
In Batch Operation

Specifies where to output an error at batch operation.

Error Output Level

Sets the output type when several errors occur.

12.10 Access Size



It is a function to set access size when the debugger accesses memory. When this setting is not done, the debugger does memory access by a command qualifiers or the most suitable size. Because it is set automatically about a built-in resource, setting is unnecessary by this function. However, the debugger does memory access by byte size on FILL, MOVE, COMPARE commands.

Start Address

Specifies the start address to be set.

End Address

Specifies the end address to be set.

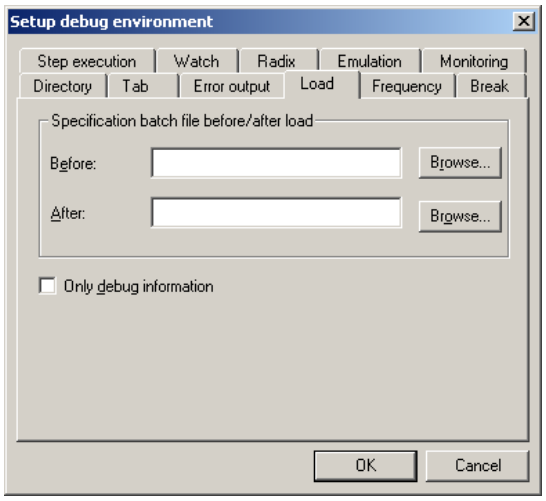
Access Size

Specifies the access size to be set. (Byte/Halfword/Word)

List

Displays the currently set area.

12.11 Load



This sets the environment when loading a target file registered in the project.

Specification Batch File before/after load

- Before

  This specifies the batch file to execute prior to the loading of the target file. This can also be changed using the Debugger's setup wizard.

- After

  This specifies the batch file to execute after the loading of the target file. This can also be changed using the debugger's setup wizard.

- Debug Information Only

  This specifies whether or not to load debug information. When checked, only the debug information is loaded.

  Consider a system where MCU application program is stored in external flash and MCU is to be started in external vector fetch mode.

  For the emulation of such a system one can connect the target with this check box marked. By doing this only debug information is loaded and with the help of Softune one can debug the application program. The restriction in this scenario is one can use only hardware breakpoint for debugging.

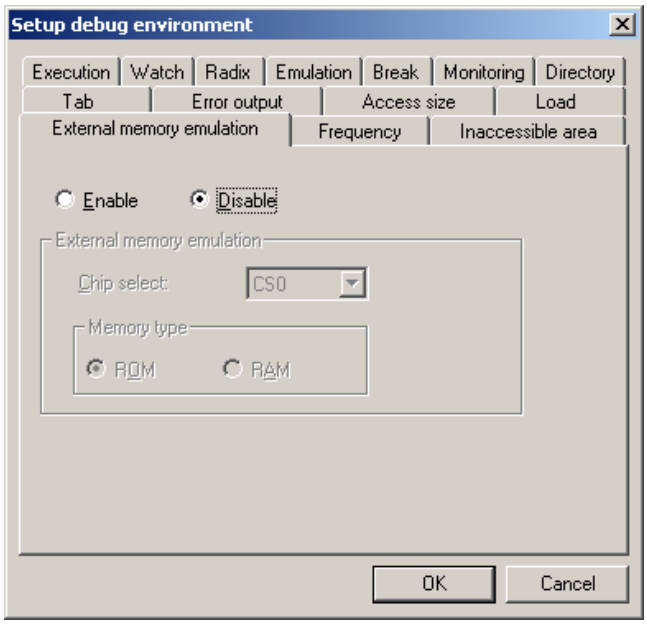
- Auto Mapping

  This specifies whether or not to enable the auto-map setting. When checked, auto-map setting is enabled.

- On demand load

  Set whether to on demand load debug information. When a checkmark is placed in the checkbox, debug information is on demand loaded.

12.12 External Memory Emulation



Enable/disable

Whether to enable or disable the external memory emulation function is specified.

Chip select

The chip select number that can be output to the external bus is specified.

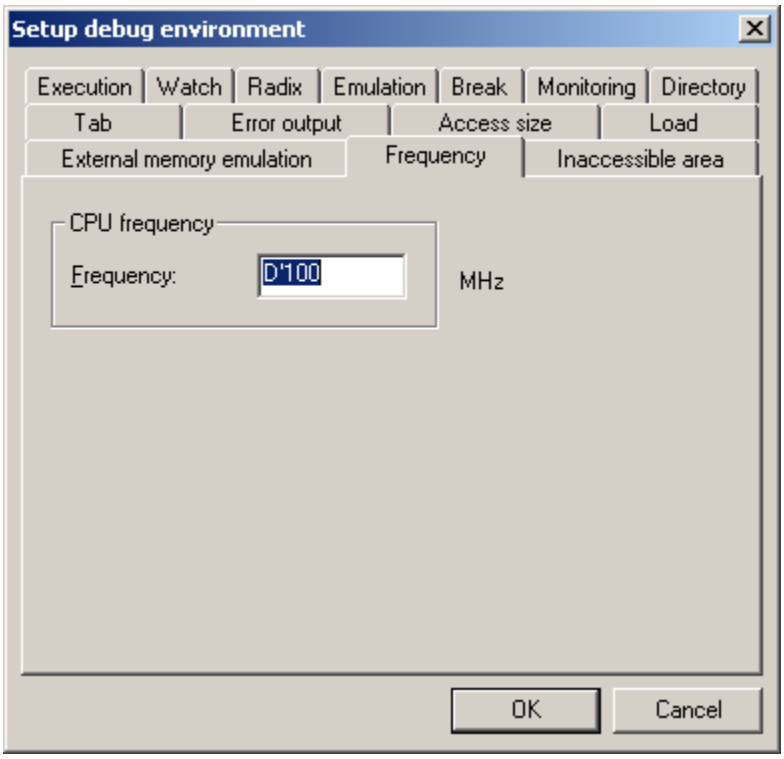
FR system: CS1 to CS5

FRex system: CS0 to CS7

Memory type

Whether to allow or inhibit write access to external memory is specified.

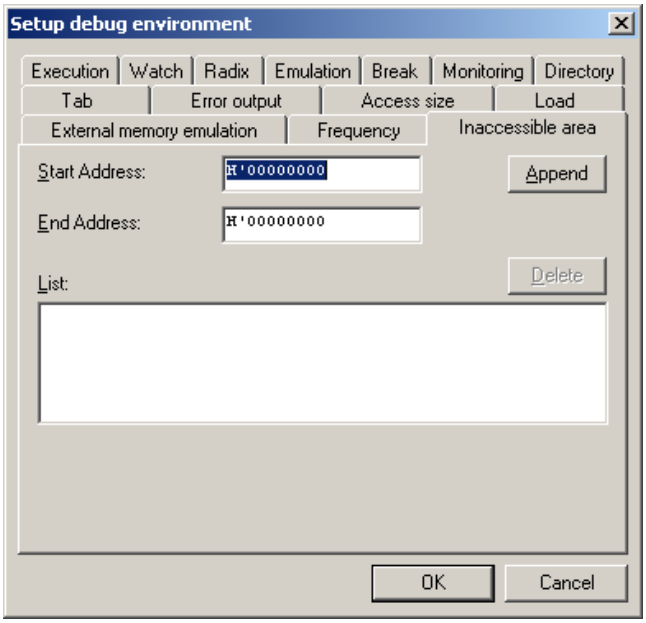
12.13 Frequency



CPU Frequency

Set maximum CPU frequency. (Lower value results in higher responsiveness of emulation system.)

12.14 Inaccessible area



Selects default code break point (i.e. Software or Hardware)

This function inhibits access to debugger memory. Up to 16 areas can be set (by increments of one byte).

Start Address

Specifies the start address to be set.

End Address

Specifies the end address to be set.

List

Displays an regions being currently set. When the check mark of the area is removed, that the area is invalidated.

13. Trigger-Input and Emulator-Output

How to Use the Two BNC Connectors

13.1 The BNC Connectors

The MB2198-01 emulator has two BNC connectors. The left one is the "TRIG"-Input and the right one the "EMUL"-Output.

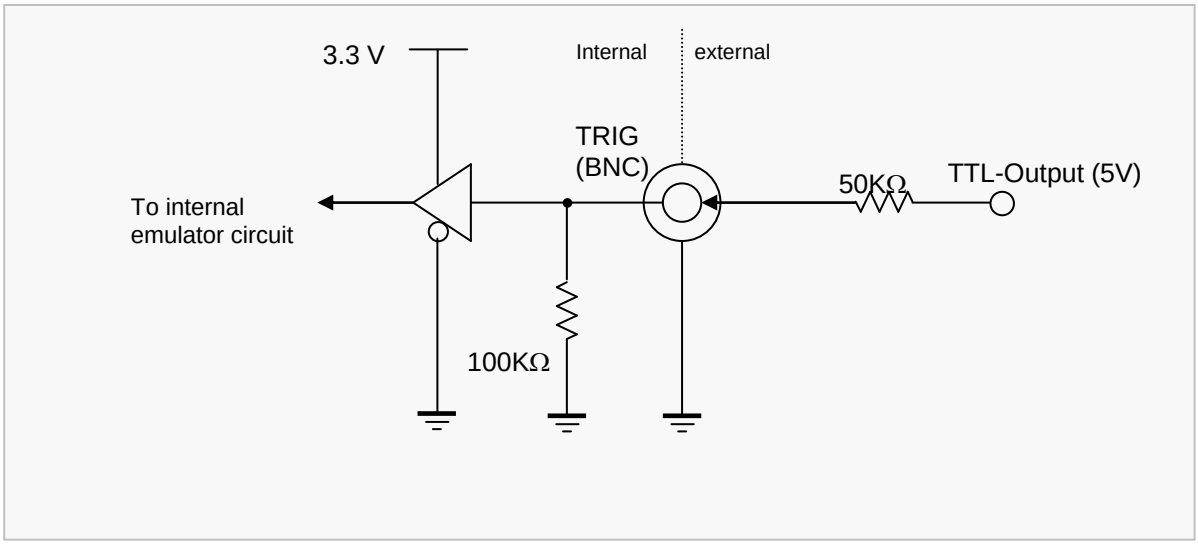
13.2 Trigger-Input

With this input an execution can be stopped. The Trigger-Input is a hardware "break point".

A logical "high" (= 3.3V) on this input stops the execution in the debugging mode. Note, that because of internal latches and different clock speeds of the emulator and the MCU the termination is not immediately. The break slip is in a range of dozens to hundreds machine clock cycles.

The execution can be resumed after a triggered break.

Because of the 3.3V input and an internal 100K pull down resistor, it is recommended to use a serial 50K resistor, if a 5V signal is used:



13.3 Emulator-Output

The BNC-Output “EMUL” goes logical “high” (= 3.3V) if a program is executed and is “low” (= 0V) if the program is stopped or a break point has occurred. This signal can be used for controlling external hardware.

14. Installing LAN

This Chapter Describes How to Install the LAN Interface

14.1 Overview

The emulator MB2198-01 are provided with a local area network adapter that can be used to program and debug a device over a network connection. No additional hardware connection except the LAN connection itself is required for this purpose

Using the Cypress LAN-remote controlled debug facility,

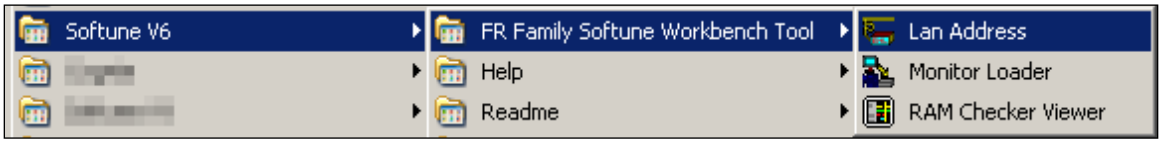
- A Cypress support engineer can easily help solving a concrete problem by debugging your application out of a Cypress support centre.
- You can control the emulator from different locations without having to move your hardware installation from one place to another.
- Program download will be more than 6 times faster than with RS232C

14.2 Configuring the LAN Adapter

1. Refer to the LAN Installation Manual coming with the LAN Adapter.
   This Application Note will give only some additional information.

2 Connect emulator and PC by the serial RS232-Port (or USB-Port)

3 Start the „LAN Address“ Program in the Softune Workbench Folder



4 Click on *Communication* to define the serial RS232 Port where the emulator is connected to or use USB connection.

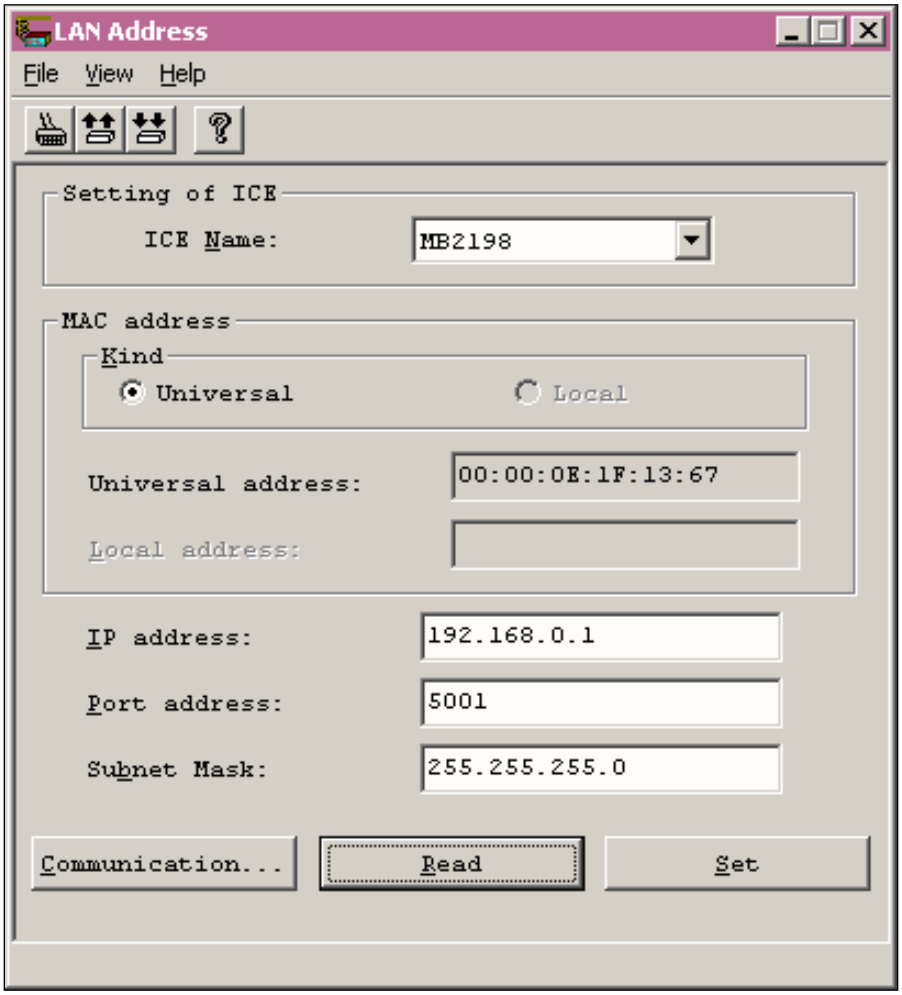
5 Read current status from the emulator

6 Set the unique IP address given from your network manager. This is an very important point, every IP-address within a network has to be unique.

7 Check the Port address: must be 5001

8 Make settings valid by „Set“, and reset the emulator when prompted

9 Close Program „LAN Address“



14.3 Configuring Operating System "Windows™"

1. Within your Windows™-directory (e.g. *C:\windows* or *C:\WINNT\system32\drivers\etc*) you should find three files:

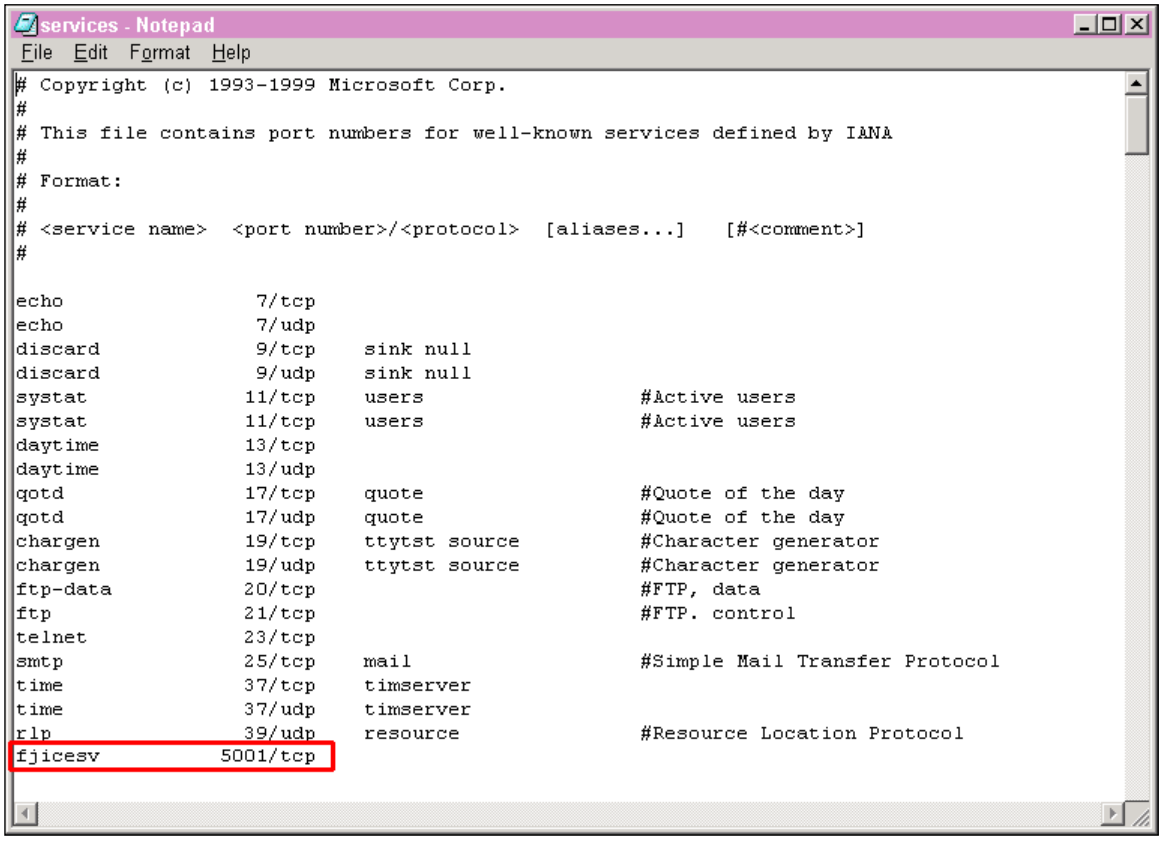
- *Services*
- *Hosts*
- *lmhosts[.sam]*

(Note: The files may not have a file-extension!)

2. Make a copy off all three files, like for an example: *services* to *sevices.old* , etc.

   Edit file *services* and add the following line:

```
fjicesv      5001/tcp            # Cypress emulator
```



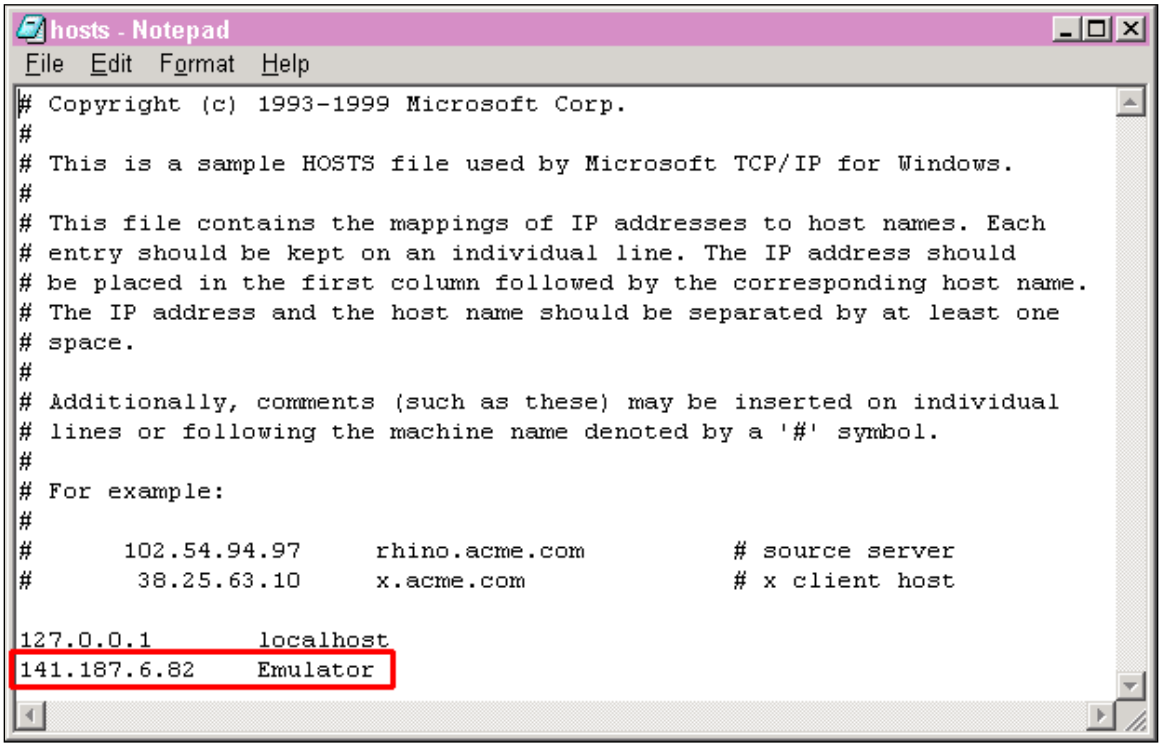
(note: if `5001/tcp` is already contained in the file s*ervices*, use an unused number beginning with 5002 or greater, e.g. `fjicesv 5002/tcp`. In that case also the emulator address given by the program „LAN address“ (see above) has to be changed!

3. When saving again the file S*ervices* be sure that your editor (e.g. notepad) will not add any extension, e.g. .txt, to your file. To get sure, use quotation marks for the filename:
File save as: **”***services***”**

4. The file *hosts* is used to make a redefinition of the complex IP-number with a simple name within your global network. This may be important if you use a DNS-Server. Of course, you can define different names for the same IP-Address, as shown below. Edit file *hosts* and add the following line:

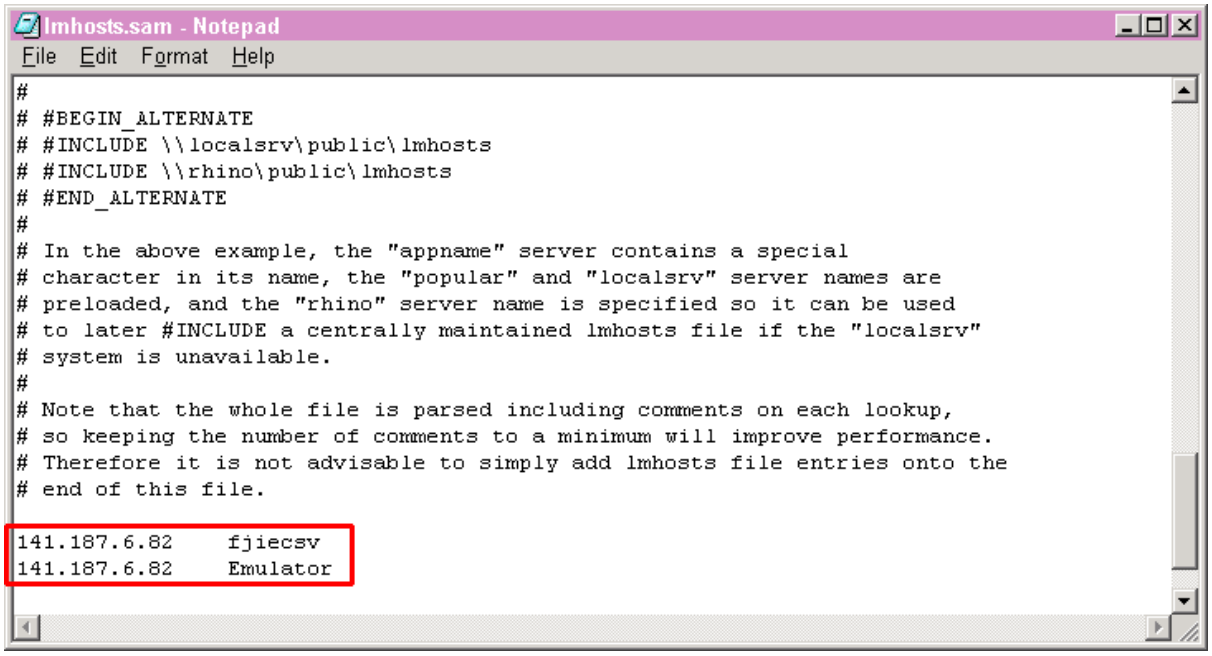
   “The unique IP address (as set by LAN-Address, see above)” “Nickname of emulator”

   e.g. 141.187.6.82 Emulator



When saving again the file *Hosts* be sure that your editor (e.g. notepad) will not add any extension, e.g. *.txt*, to your file. To get sure, use quotation marks for the filename:
File save as: ”*hosts*”

5. The file *lmhosts* (or *lmhosts.sam*) is used to make a redefinition of the complex IP-number with a simple name within your local network.
Of course, you can define different names for the same IP-Address, as shown below. Edit file *lmhosts* and add the following line:



"the unique IP address (as set by LAN-Address, see above)" "nickname of emulator"
e.g.

```
141.187.6.53      fjicescv
141.187.6.53      Emulator
```

When saving again the file *Lmhosts* get sure that your editor (e.g. notepad) will not add any extension, e.g. .txt, to your file. To get sure, use quotation marks for the filename:
e.g. File save as: **"***lmhosts***"** (or *lmhosts.sam*)

14.4 Checking the network-connection

Disconnect serial RS232 (or USB) cable from Emulator and try to find the emulator:

- Open DOS-Window (or open RUN (Ausführen) in the Start-Menu)
- The command `ping Emulator` should acknowledge with some time-values, that means the network is set up right.

The emulator with LAN-adapter is successfully integrated in the network environment and can be used by the Softune Workbench.

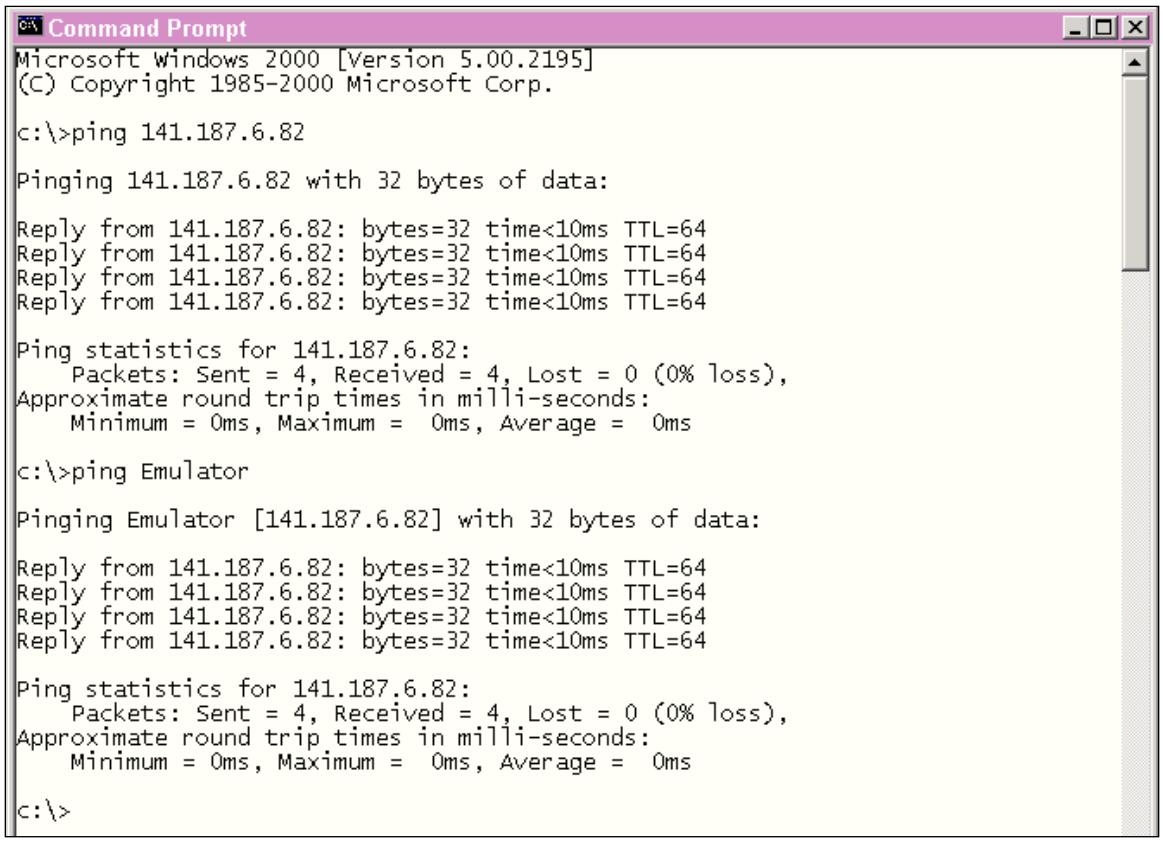
14.5 Troubleshooting

If the command `ping my_emulator` will reply with a timeout-message, try to find the emulator by its IP-address: Type command `ping` *`IP-address`*, e.g. `ping 141.187.6.53`.

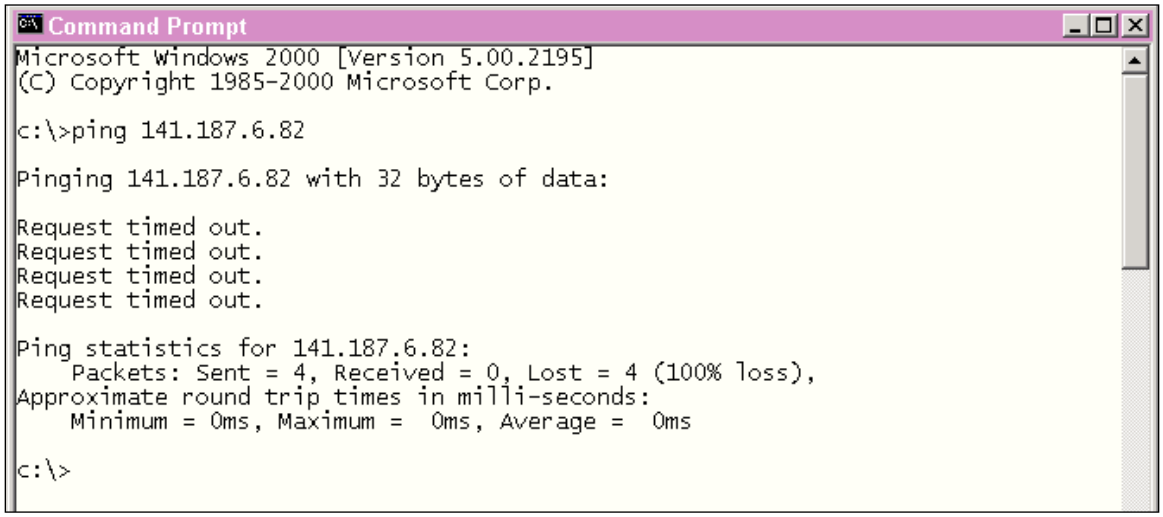
If this will work, then check settings (nickname and IP-address) within *Lmhosts* and *Hosts*.
If neither nickname (`my_emulator`) nor IP-address will work, check settings done by the program *LAN-address* and check the file *Services*.

Also check your physical network interconnection cables. Please keep in mind, that the emulator only works with 10Mbit/s. This means in case that for a 100Mbit/s network a 100Mbits/10Mbit - HUB is needed. When using a HUB for 10Base-T then a standard (1:1) network cable has to be used. If the emulator is connected directly to the PC a „crossed network cable“ is necessary.

Good answer: Emulator/LAN-adapter replies with time-values:

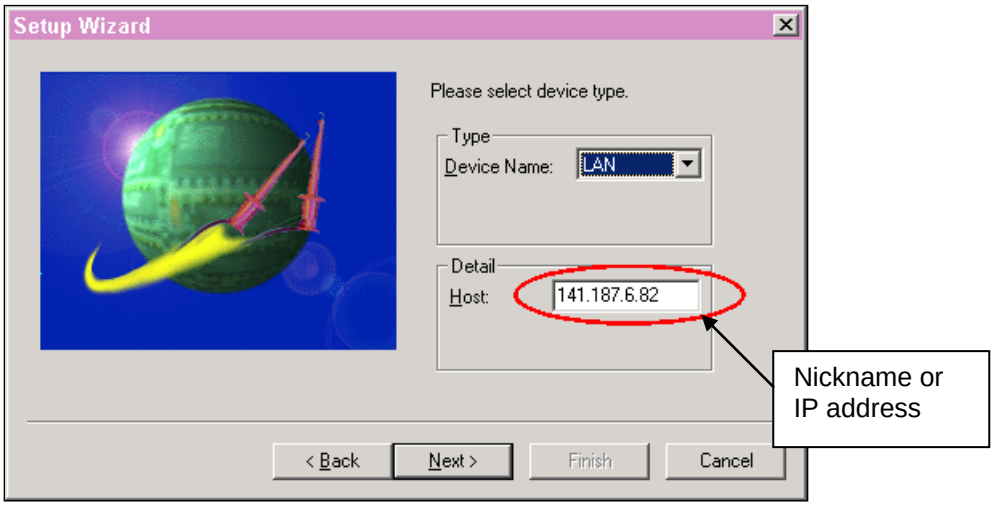


"Failed"-answer: Emulator/LAN-adapter replies with timeout message:



14.6 Softune Workbench

Within the Softune Workbench the LAN interface can be used instead of a serial RS232C communication. Detail-Host: can be the „nickname“ as defined in the files *hosts* and *lmhosts* or the IP-address of the emulator, set by the program „LAN-Address“, can be used.



15. Miscellaneous

Remarks and Hints

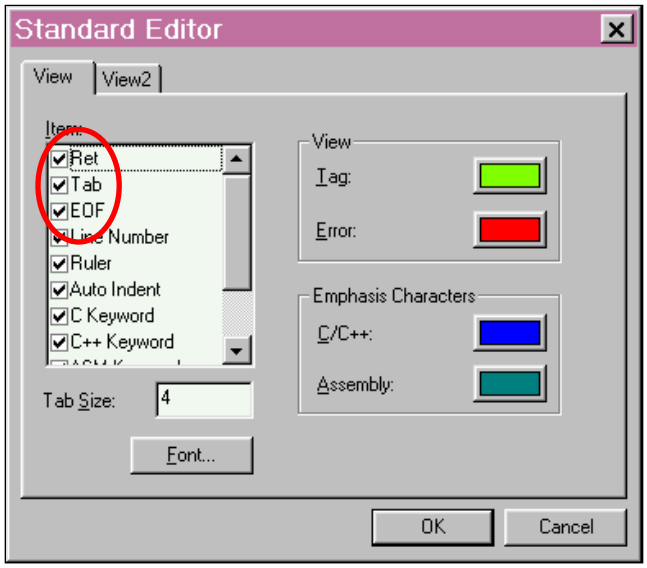
15.1 View Mode of the Editor

When starting the Softune Workbench Software for the first time, the text editor has some default viewing settings, which can be switched off. Source codes will look like in the left picture.

```
21 unsigned char LED;↓
22 ↓
23 /* Sub Routines */↓
24 ↓
25 void init_timer(void)↓
26 {↓
27 ^    TMRLR0 = 0x61A7;^    ^    /* relo
28 ^    TMCSR0 = 0x81B;^^    ^    /* pres
29 }↓
```

You can disable the viewing of the tabulators, Return signs and the End-of-File delimiter by clicking on the right mouse button just over the text window. Then a large pop up window will open.

Choose *Customize*. Then the following new window will occur:



The first three entries in the item list are for selecting and deselecting the view of these non-printing characters.

16. Appendix

16.1 Related Documents

Please find further information in the following documents.

- MB2198-01 Hardware Manual (Emulator)
- MB2198-01 Getting Started Application Note
- MB2198-01 Installation Guide Application Note

17. Additional Information

Information about Cypress Microcontrollers can be found on the following Internet page:

http://www.cypress.com/cypress-microcontrollers

Revision History

Document Revision History

Document Title: FR Family MB2198-01 Emulator System Getting Started Guide Document Number: 002-05222			
Revision	**Issue Date**	**Origin of Change**	**Description of Change**
**	11/13/2003	NOFL	Initial release
	01/25/2008		Installing LAN chapter added
	07/09/2008		Complete reorganisation and based on MB91460 Series
*A	03/22/2016	NOFL	Migrated Spansion Guide from MCU-AN-391027-E-V20 to Cypress format